CDC® CYBER 200 OPERATING SYSTEM VERSION 1

FOR USE WITH
CYBER 200 SERIES
COMPUTER SYSTEM

Volume 2 of 2

REFERENCE MANUAL

CONTROL DATA

60457010

CDC® CYBER 200 OPERATING SYSTEM VERSION 1

FOR USE WITH
CYBER 200 SERIES
COMPUTER SYSTEM

Volume 2 of 2

REFERENCE MANUAL

CONTROL DATA

60457010

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(07-02-79)	
B	Revised to update manual to CYBER 200 Operating System 1.5 and to make editorial and technical
(08-22-80)	corrections. New features documented include CYBER 200 Model 205 support, service station elimina-
	tion, abnormal termination control (ATC), job duration files enhancement, and job dayfile enhance-
	ment. This edition obsoletes all previous editions.
C	Revised to update manual to CYBER 200 Operating System 1.5.1 and to make editorial and technical
(12-05-80)	corrections. This revision documents input queue manager support including the EXECUTE IQM REQUEST and
	PROCESS SYSTEM PARAMETER system messages.

Publication No.
60457010

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

© 1979, 1980
by Control Data Corporation
All rights reserved
Printed in the United States of America

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Title Page	-
ii	C
iii/iv	C
v/vi	C
vii	C
viii	C
ix	C
x	C
1-1	C
1-2	C
1-3	C
1-4	C
2-1	C
2-2	C
2-3	C
2-4	C
2-5	C
2-6	C
2-7	C
2-8	C
2-9	C
2-10	C
2-11	C
2-12	C
2-13	C
2-14	C
2-15	C
2-16	C
3-1	C
3-2	C
3-3	C
4-1	C
4-2	C
4-2.1/ 4-2.2	C
4-3	B
4-4	B
4-5	B
4-6	B
4-7	B
4-8	B
4-8.1/ 4-8.2	C
4-9	B
4-10	B
4-11	B
4-12	C
4-13	C
4-14	B
4-15	C
4-16	B
4-17	B
4-18	B
4-19	B
4-20	B
4-21	B
4-22	C
4-22.1/ 4-22.2	C
4-23	C
4-24	C
4-24.1/ 4-24.2	C
4-25	B
4-26	B
4-27	B
4-28	B
4-29	B
4-30	B
4-31	C
4-32	C
4-33	C
4-34	C
4-35	C
4-36	B
4-37	B
4-38	B
4-39	C
4-40	C
4-41	B
4-42	B
4-43	B
4-44	B
4-45	B
4-46	B
4-47	B
4-48	B
4-49	B
4-50	B
4-51	C
4-52	C
4-53	C
4-54	B
4-55	B
4-56	C
4-57	B
4-58	C
4-58.1/ 4-58. 2	C
4-59	C
4-60	C
4-61	C
4-62	C
4-63	C
4-64	C
4-65	C
4-66	C
4-67	C
4-68	C
4-69	C
4-70	C
4-70.1/ 4-70.2	C
4-70.3/ 4-70.4	C
4-70.5/ 4-70.6	C
4-70.7/ 4-70.8	C
4-71	C
4-72	C
4-73	C
4-74	C
4-75	C
4-76	C
4-77	C
4-78	C
4-79	C
4-80	C
4-81	C
4-82	C
4-83	C
4-84	C
4-85	C
4-86	C
4-87	C
4-88	C
4-88.1	C
4-88.2	C
4-89	C
4-90	C
4-91	C
4-92	C
4-93	C
4-94	C
4-95	C
4-96	C
4-97	C
4-98	C
4-98.1/ 4-98.2	C
5-1	C
5-2	A
5-3	A
5-4	A
5-5	A
6-1	A
7-1	C
7-2	C
7-2.1/ 7-2.2	C
7-3	A
7-4	A
7-5	C
7-6	C
7-7	C
7-8	B
7-9	C
7-10	C
8-1	B
8-2	B
8-3	B
8-4	B
8-5	B
8-6	B
8-7	B
8-8	B
8-9	B
8-10	C
8-11	B
8-12	C
9-1	A
9-2	A
9-3	A
9-4	A
9-5	A
9-6	A
9-7	A
9-8	A
9-9	A
9-10	A
9-11	A
A-1	A
A-2	A
A-3	C
A-4	C
B-1	B
B-2	B
B-3	B
B-4	B
B-5	B
B-6	B
B-7	B
B-8	B
B-9	C
B-10	A
B-11	A
B-12	A
B-13	A
B-14	B
B-15	C
B-16	C
C-1	B
C-2	B
C-3	B
C-4	B
D-1	B
D-2	A
D-3	B
D-4	B
D-5	B
E-1	B
E-2	B
E-3	B
F-1	A
Index-1	B
Index-2	B
Index-3	B
Index-4	B
Comment Sheet	C
Back Cover	-

PREFACE

This manual describes the CONTROL DATA® CYBER 200 Operating System for the CDC® CYBER 200 Computer System. The CYBER 200 configuration supports either a CYBER 200 Link Station running under control of the NOS/BE 1 or SCOPE 3.4 Operating System, or a CYBER 200 Access Station running under control of the NOS 1 Operating System.

This manual is published in two volumes:

- Volume 1 is written for applications programmers who will be accessing CYBER 200 Operating System capabilities through batch jobs or interactive terminals. A detailed knowledge of basic programming principles is assumed. Subroutines that interface with the Operating System through CYBER 200 FORTRAN calls or CYBER 200 Assembler language calls are included. File concepts and debugging information are also presented.
- Volume 2 is written for systems programmers who will be modifying or expanding the capabilities of the CYBER 200 Operating System, and who wish to do so by writing programs that communicate directly with other programs and with the virtual system part of the operating system. All virtual system messages available for writing such programs, as well as some of the system tables referred to by these messages, are described. It is assumed that the reader has some understanding of the principles of operating systems in general.

RELATED PUBLICATIONS

Related information can be found in the following publications.

Control Data Publication	Publication Number
CYBER 200 Operating System Reference Manual Volume 1	60457000
CYBER 203 Computer System Hardware Reference Manual	60256010
CYBER 205 Computer System Hardware Reference Manual	60256020
CYBER 200 Operating System Operator's Guide	60457030
CYBER 200 Operating System Installation Handbook	60457020
CYBER 200 Access Station Reference Manual	60452800
CYBER 200 Link Reference Manual	60457060
CYBER 200 Link Operator's Guide	60457070
CYBER 200 FORTRAN Language Reference Manual	60457040
CYBER 200 Assembler Reference Manual	60457050

Control Data manuals can be ordered from:

Literature and Distribution Services
STP005
308 North Dale Street
St. Paul, Minnesota 55103

DISCLAIMER

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

1. INTRODUCTION TO CYBER 200 1-1

Central Operating System 1-1
Resident System 1-1
Virtual System 1-1
Queuing of Virtual System Tasks 1-1
Scheduler Interaction with PAGER 1-2
Privileged System Tasks 1-2
Obtaining Privileged Status 1-2
Privileged Resident System Calls 1-3
Virtual System Calls 1-3
Peripheral Operating System 1-4

2. JOB MANAGEMENT TABLES 2-1

File Index Table 2-1
Minus Page 2-7
Input/Output Connectors 2-7
Explicit Input/Output Information 2-7
Map Directories 2-7
Interrupt Address Stack 2-8
Minus Page File Maps 2-12
Bound Explicit Maps 2-12
Bound Implicit Maps 2-12
Drop File Map 2-12

3. FILE CONCEPTS 3-1

File Names 3-1
File Ownership 3-1
File Access and Lockout 3-1
File Management Categories 3-2
Mass Storage Files 3-2
Scratch Files 3-2
Output Files 3-2
Drop Files 3-2
Write-Temporary Files 3-2
File I/O 3-2
Physical Files 3-2
Virtual Files 3-3
Magnetic Tape Files 3-3

4. SYSTEM MESSAGES 4-1

System Message Execution 4-1
Alpha and Beta Word Conventions 4-1
Overview 4-1
Messages 4-2
Create File (f=0001) 4-3
Destroy File (f=0002) 4-9
Open File (f=0003) 4-10
Map (f=0004) 4-16
Close File (f=0005) 4-18
Terminate (f=0006) 4-21
List Unformatted File Index Table (f=0007) 4-22
Give File (f=0008) 4-23
List File Index or System Table (f=0009) 4-25
Reduce File Length (f=000A) 4-31
Change File Attributes (f=000B) 4-31
Give Tape Access to Controllee (f=000C) 4-35
Route and File Disposition (f=000D) 4-36
User/Accounting Communication (f=000E) 4-38
Attach Permanent File (f=0010) 4-40
Get Pack Label and PFI (f=0011) 4-40
List Controllee Chain (f=0013) 4-42
Send a Message to Controller (f=0014) 4-44
Send a Message to Controllee (f=0015) 4-44
Get Message from Controller or Operator (f=0016) 4-47
Get Message from Controllee (f=0017) 4-48
Message Control (f=0018) 4-50
Remove Controllee from Main Memory (f=0019) 4-50
Send a Message to Operator (f=001A) 4-51
Initialize or Disconnect Controllee (f=001B) 4-52
Program Interrupt (f=001C) 4-54
Initialize Controllee Chain (f=001D) 4-55
Enable/Disable Abnormal Termination Control (f=0020) 4-56
Execute Operator Command (f=0021) 4-58
Execute Program for User Number (f=0022) 4-70.7
Update User Directory (f=0023) 4-71
Miscellaneous (f=0024) 4-76
Recall (f=0025) 4-79
Pool File Manager (f=0026) 4-79
Link (f=0027) 4-80
Variable Rate Accounting (f=0028) 4-86
Send Message to Dayfile (f=0029) 4-86
Execute IQM Request (f=0030) 4-88
Explicit Input/Output (f=0050) 4-88.2
Return from Interrupt (f=0051) 4-94
Give Up CPU Until I/O Completes (f=0052) 4-94
Advise (f=FF00) 4-94
Process System Parameter (f=FF01) 4-98

5. VIRTUAL SYSTEM DEBUG TOOL 5-1

Resident System 5-1
Virtual System 5-1
User and System Interfaces 5-1
Shared Table 5-1
Commands 5-1
Command Format 5-1
Debug Commands 5-3
Control Commands 5-4
Error Messages 5-5

6. ANALYZER 6-1

7. ACCOUNTING 7-1

Calculation of STUs 7-1
Repositories 7-1
Statistics Accumulation 7-2
Cumulative Accounting Buffer 7-2
Accounting File 7-2
Active Accounting File Blocks 7-3
Task Records 7-4
Terminal Records 7-4

Disk File Management Records 7-4
Tape Records 7-8
System Records 7-9
Job Records 7-9
Standardized Accounting Enhancements 7-9
Calculation of SBUs 7-9
Variable Rate Accounting 7-9
Variable Rate Table 7-9
Virtual System Table Definition 7-10
File Maintenance 7-10

8. COMMON EXECUTE LINE SUPPORTING ROUTINES 8-1

Conventions 8-1
Supporting Routines 8-2
Q7MODE 8-3
Q7PROMPT 8-3
Q7KEYWRD 8-3
lhs Table 8-4
rhs Table 8-4
Return Buffer 8-9
Special Characters 8-12

9. LOADER CONVENTIONS 9-1

General Table Structure 9-1
Module Tables 9-2
Module Header Table 9-2
Code Block Table 9-2
Code Relocation Table 9-3
External/Entry Table 9-3
Interpretive Data Initialization Table 9-5
Interpretive Relocation Initialization Table 9-9
Transfer Symbol Table 9-9
Debug Symbol Table 9-9
Symbol Definition Table 9-9
Pseudo Address Vector Table 9-9

APPENDIXES

A. CHARACTER SET A-1
B. DIAGNOSTICS B-1
C. GLOSSARY C-1
D. REGISTER FILE CONVENTIONS D-1
E. CYBER 200 OPERATING SYSTEM INVISIBLE PACKAGE E-1
F. PROGRAM STATES F-1

INDEX

FIGURES

2-1 File Index Table Format 2-2
2-2 Format of Input/Output Connector for a Mass Storage File Opened for Explicit Input/Output 2-9
2-3 Format of Input/Output Connector for Implicit Input and Output 2-11
2-4 Format of Input/Output Connector for Tape and Link Files 2-13
2-5 Format of Word #97 of the Minus Page 2-14
2-6 Map Directory Format 2-14
2-7 Interrupt Address Stack Format 2-14
2-8 Bound Explicit Map Entry Format 2-15
2-9 Bound Implicit Map Entry Format 2-15
2-10 Drop File Map Entry Format 2-16
2-11 Drop File Map Format 2-16
4-1 CREATE FILE Message Format 4-3
4-2 DESTROY FILE Message Format 4-8
4-3 Format of Bound Implicit Map Entry in Program Buffer 4-11
4-4 OPEN FILE Message Format 4-12
4-5 MAP Message Format 4-16
4-6 CLOSE FILE Message Format 4-19
4-7 TERMINATE Message Format 4-21
4-8 LIST UNFORMATTED FILE INDEX TABLE (f=0007) 4-22
4-9 GIVE FILE Message Format 4-23
4-10 LIST FILE INDEX OR SYSTEM TABLE Message Format 4-26
4-11 REDUCE FILE LENGTH Message Format 4-31
4-12 CHANGE FILE NAME OR ACCOUNT Message Format 4-32
4-13 GIVE TAPE ACCESS TO CONTROLLEE Message Format 4-35
4-14 ROUTE AND FILE DISPOSITION Message Format 4-36
4-15 USER/ACCOUNTING INFORMATION Message Format 4-39
4-16 Attach Permanent File Message Format 4-40
4-17 GET PACK LABEL AND PFI Message Format 4-41
4-18 LIST CONTROLLEE CHAIN Message Format 4-43
4-19 SEND A MESSAGE TO CONTROLLER Message Format 4-45
4-20 SEND A MESSAGE TO CONTROLLEE Message Format 4-46
4-21 GET MESSAGE FROM CONTROLLER OR OPERATOR Message Format 4-47
4-22 GET MESSAGE FROM CONTROLLEE Message Format 4-48
4-23 MESSAGE CONTROL Message Format 4-50
4-24 REMOVE CONTROLLEE FROM MAIN MEMORY Message Format 4-51
4-25 SEND A MESSAGE TO OPERATOR Message Format 4-52
4-26 INITIALIZE OR DISCONNECT CONTROLLEE Message Format 4-53
4-27 PROGRAM INTERRUPT Message Format 4-54
4-28 INITIALIZE CONTROLLEE CHAIN Message Format 4-55
4-29 ENABLE/DISABLE ABNORMAL TERMINATION CONTROL Message Format 4-57
4-30 EXECUTE OPERATOR COMMAND Message (Alpha) Format 4-58
4-31 EXECUTE OPERATOR COMMAND Message (Beta) Format 4-59
4-32 Input Queue Entry Format 4-70.5
4-33 T_JCAT System Table Format 4-70.6
4-34 EXECUTE PROGRAM FOR USER NUMBER Message Format 4-70.7

4-35 UPDATE USER DIRECTORY Message (Alpha) Format 4-71
4-36 UPDATE USER DIRECTORY Message (Beta) Format 4-72
4-37 MISCELLANEOUS Message Format 4-76
4-38 RECALL Message Format 4-79
4-39 POOL FILE MANAGER Message Format 4-80
4-40 LINK Message (Alpha) Format 4-82
4-41 VARIABLE RATE ACCOUNTING Message 4-86
4-42 SEND MESSAGE TO DAYFILE Message Format 4-87
4-43 EXECUTE IQM REQUEST (f=0030) 4-88
4-44 EXPLICIT INPUT/OUTPUT Message (Alpha) Format 4-89
4-45 EXPLICIT INPUT/OUTPUT MESSAGE (Beta) Format 4-90
4-46 RETURN FROM INTERRUPT Message Format 4-95
4-47 GIVE UP CPU UNTIL I/O COMPLETES Message Format 4-96
4-48 ADVISE Message Format 4-96
4-49 PROCESS SYSTEM PARAMETER (f=FF01) 4-98
6-1 ANALYZER Execute Line Format 6-1
7-1 Repository Table Entry Format 7-1
7-2 Master Clock Format 7-3
7-3 Master Clock Example 7-3
7-4 General Format of Accounting Record First Word 7-5
7-5 Task Record Header Format 7-5
7-6 Task Record Format, Subtype 0 7-6
7-7 Task Record Format, Subtype 2 7-7
7-8 Terminal Record Format, Subtype 0 7-8
7-9 Terminal Record Format, Subtypes 1 and 3 7-8
7-10 Algorithm for SBU Calculation 7-9
8-1 Key-Dependent Parameter Format 8-2
8-2 Q7MODE Call Statement Parameters 8-3
8-3 Q7PROMPT Call Statement Parameters 8-3
8-4 Q7KEYWRD Call Statement Parameters 8-5
8-5 lhs Table Pointer Configuration 8-5
8-6 lhs Table Format 8-5
8-7 lhs Table Header Format 8-6
8-8 lhs Table Entry Format 8-6
8-9 rhs Table Format 8-7
8-10 General Format for First Word of rhs Table Entry 8-7
8-11 rhs Table Entry Format, Type 2 8-8
8-12 rhs Table Entry Format, Type 6 8-8
8-13 Return Buffer Format 8-9
8-14 General Format for First Word of Return Buffer Entry 8-9
8-15 Return Buffer Entry Format, Types 1 and 2 8-9
8-16 Return Buffer Entry Format, Types 4 and 5 8-9
8-17 Return Buffer Entry Format, Type 6 8-10
8-18 Return Buffer Entry Format, Type 7 with Zeroed Flags 8-10
8-19 Return Buffer Entry Format,Type 7 with Set Flag 8-11
8-20 Return Buffer Entry Format, Type 8 with One Set Flag 8-11
8-21 Return Buffer Entry Format, Type 8 with Two Set Flags 8-12
9-1 Loader Table Header Format 9-1
9-2 Module Header Table Format 9-2
9-3 Code Relocation Table Format 9-3
9-4 External/Entry Table Format 9-4
9-5 Descriptor Format for Externals and Entry Points 9-5
9-6 Data Item Format 1 9-5
9-7 Data Item Format 2 9-6
9-8 Data Item Format 3 9-7
9-9 Type D Data Item Format 9-8
9-10 Debug Symbol Table Format 9-9
9-11 Symbol Definition Table Entry Format 9-10
9-12 Pseudo Address Vector Table Entry Formats 9-11

TABLES

1-1 Resident System Calls for Privileged Users 1-3
1-2 Virtual System Calls Available to Privileged User Tasks Only 1-4
1-3 Virtual System Calls with Options Available to Privileged User Tasks Only 1-4
2-1 File Disposition Specifications in File Index Table 2-1
2-2 File Characteristics Specifications in File Index Table 2-1
2-3 Minus Page Format 2-8
3-1 File Index Table Fields That Affect File Ownership 3-1
4-1 Message Function Codes 4-2
4-2 Parameters for Link File Create 4-8.1
4-3 Appropriate Error Codes for Input/Output Operations 4-93
5-1 Structure of the T_VSD Table 5-2
5-2 VSDT Command Summary 5-3
5-3 VSDT Commands for Setting Breakpoints 5-4
5-4 VSDT Command for Resetting Breakpoints 5-4
5-5 VSDT Command for Accessing Paged-Out Addresses 5-5
5-6 VSDT Error Messages 5-5
7-1 Accounting Record Type and Subtype Codes 7-3
7-2 Active Accounting File Block Format 7-4
7-3 Format of First Block of Active Accounting File 7-5
7-4 System Resources 7-10
8-1 Execute Line Special Characters 8-12
9-1 Module Header Table Types 9-3

NOTATIONS USED IN THIS MANUAL

UPPERCASE	Specific symbolic names of entities such as system variables and system procedures. When a string of uppercase letters appears in a call, the letters must be entered as shown.
lowercase	Generic names that represent specific names and parameters supplied by the programmer.
.	Generic names are frequently subscripted in this manual to distinguish the separate members of the general class.
. . .	Repetition indicator, used in this manual both horizontally and vertically. The portion of the format immediately preceding (or immediately above or below) can be repeated at programmer option, or has been omitted for the sake of brevity.
△	Blank indicator. In a character string, this symbol indicates that a blank character or a space should appear.
#	All numbers used in this manual are decimal unless they are denoted as hexadecimal. All hexadecimal numbers are prefixed by the # character.

Punctuation characters shown within a format are required unless the text indicates another punctuation character can be substituted.

The CYBER 200 Operating System (CYBER 200 OS) consists of a central operating system, which runs in the central processing unit (CPU), and a peripheral operating system, which runs in the station processors. Each operating system consists of a resident system and a nonresident set of callable tasks.

CENTRAL OPERATING SYSTEM

The central operating system can be divided into three parts:

- The resident system runs in a hardware mode called monitor mode. It is always resident in core and references memory by absolute address rather than through the virtual paging mechanisms. When the resident system is running, interrupts are inhibited and some extra instructions are enabled.
- The virtual system runs in a hardware mode called job mode. It consists of a nonresident set of tasks that perform such functions as controlling entry of users into the system, file management, explicit input/output, and terminal message handling. Virtual system tasks communicate with the resident system by using privileged resident system calls. The virtual system can modify system tables directly.
- Privileged system tasks run in the hardware mode called job mode and perform many of the same functions as virtual system tasks. Privileged system tasks can issue privileged resident system calls to communicate with the resident system. Privileged system tasks cannot, however, modify system tables directly.

CYBER 200 hardware modes are described in the CYBER 200 Hardware Reference Manual.

RESIDENT SYSTEM

The resident portion of the central operating system contains:

- KERNEL, which handles alternator management (time-slicing) and message communication.
- PAGER, which is responsible for main memory allocation and page swapping.

All communications between the various portions of the system are by means of system messages. These messages either pass through KERNEL, which in this case acts as a message switcher, or are processed directly by KERNEL. User jobs, privileged tasks, and virtual system tasks communicate messages to KERNEL through the exit force instruction (a machine language instruction). PAGER communicates messages to KERNEL by direct calls. The peripheral system communicates with KERNEL by setting pointers in the station queuing structure; KERNEL communicates with the peripheral system by setting pointers and setting station channel flags.

The time-slice management portion of KERNEL is controlled by an alternator loop that acts as a circulartable with various indicators in each table entry. These indicators include a pointer to a minus page table entry, a descriptor block table entry, and three sets of flag bits that define the status of each alternator loop entry.

A unique entry in the alternator loop is shared by all virtual system tasks; to prevent two routines from modifying the same system table simultaneously, only one system task is allowed to run at a time. This system alternator slot has the highest priority; it is always run unless blocked for input/output or PAGER action.

All memory access interrupts, as well as certain messages dealing with physical memory allocation, are given by KERNEL to PAGER for processing. PAGER dynamically allocates both large and small pages, and performs all required implicit input/output necessary to free memory pages and obtain the pages causing memory access interrupts.

That portion of memory that is not permanently occupied by the resident system and its tables is available for allocation to executing system and user tasks, including the virtual system. This allocatable memory is either already allocated space (space reserved for use by a specific task) or else is free space (space not allocated to any task).

VIRTUAL SYSTEM

The virtual system contains routines for system functions such as file management, input/output, message handling, and CPU scheduling.

Queuing of Virtual System Tasks

Virtual system demand tasks, which are critical to the efficient working of the operating system, are queued by one of two occurrences:

- Bits are set in one or more alternator slots to indicate that virtual system action is required.
- PAGER requests KERNEL to queue a virtual system demand task.

Periodic virtual system tasks, which are not considered critical, are queued by one of three occurrences:

- A communication from a station requires processing.
- A user job issues a message that requests a system service not provided by the resident system.
- An entry in the periodic table indicates it is time to run a virtual system task.

Scheduler Interaction with PAGER

Estimates of the memory requirements of any task in the system are based on the size of the task's working set, that is, the most frequently referenced pages. Before a task begins execution, the scheduler estimates the task's initial working set size based on the task's drop file size. PAGER then monitors the memory usage of the executing task and determines which of the task's allocated pages are still in the task's working set and which are not.

The CPU scheduler determines when a waiting task is scheduled for execution in central memory. A task qualifies for scheduling when it is the highest priority task whose working set plus memory for working set growth fits into uncommitted memory. When a task is scheduled, it is entered in the CPU scheduling list (CPUQ) and is a candidate for CPU assignment. Each task in the CPUQ can increase its working set by the number of pages reserved for growth. If more pages are needed, memory becomes overcommitted and the lowest priority task must be evicted (removed from CPUQ).

PRIVILEGED SYSTEM TASKS

Any task can be called a privileged task if it is running under a privileged user number, has the privilege flag set in the file index table entry of its source-file, is the system routine OPERATOR, or is a USER1 routine. A privileged system task is any privileged task (such as a USER1 routine) that is part of the central operating system. A privileged user task is any other privileged task, such as one that a system or installation-defined utility might use.

Privileged system tasks have basically the same characteristics as virtual system tasks in that they run in job mode, make resident calls, are pageable, and can access the files of other users. Unlike the virtual system tasks, privileged tasks do not have direct access to system tables; however, through privileged calls to the virtual system, they are able to obtain indirect access to the tables.

Because they can make most resident system calls, privileged system tasks are able to perform some functions for the virtual system. This reduces virtual system overhead and frees the virtual system to process other functions. Tasks such as handling input and output files and operator communications are currently done by privileged system tasks.

CARDREDR, IQM, PRINTOUT, PUNCHOUT, and OPERATOR are privileged system tasks that run under the privileged user numbers 000002, 000001, 000003, 999999, and 999998:

CARDREDR Transfers input files from the service station to mass storage using C3xx resident calls. If the file is a batch input deck, it gives the file to IQM; otherwise, it gives the file to the appropriate user.

IQM Controls batch job submission to the CPU scheduler based on scheduling constraints such as time limit, working set size, and large page limit.

PRINTOUT Transfers print output files from mass storage to the service station using C3xx resident calls.

PUNCHOUT Transfers punch output files from mass storage to the service station using C3xx resident calls.

OPERATOR Enables the operator to communicate with the system by issuing the EXECUTE OPERATOR COMMAND message (f=0021), enabling the operator to display memory, task, user information, and equipment status. The operator is able to assign tapes and make tape or disk units logically available or unavailable to the system. User directory modifications can also be made via this task.

POSTOUT is a privileged system task that is executed by the system only if the CYBER 200 Computer System is configured with an access station:

POSTOUT Transfers permanent files from CYBER 200 to the access station.

SMP, CYBERIN, and CYBEROUT are privileged system tasks that are executed by the system only if CYBER 200 is configured with a CYBER 200 Link Station:

SMP Processes the message protocol between the CYBER 200 Link Station and CYBER 200.

CYBERIN Transfers job files from the CYBER front-end to CYBER 200.

CYBEROUT Transfers output files from CYBER 200 to the CYBER front-end.

The privileged system tasks CARDREDR, IQM, PRINTOUT, PUNCHOUT, POSTOUT, SMP, CYBERIN, and CYBEROUT are collectively referred to as USER1 routines.

Obtaining Privileged Status

A user is called a privileged user if the privilege flag (udtrust field) in the user directory entry is set. The flag can be set by the CREATE USER operator command, the CHANGE operator command, or by an UPDATE USER DIRECTORY (f=0023) message. A user task and all controllees and controllee chains initiated by it acquire privileged status if one or both of the following are true:

- The task is running under a privileged user number.
- The privilege flag (fi field) in the source-file's file index table entry is set.

An executable file can, for example, run as a privileged task under a nonprivileged user number if the privilege flag in the source file's file index table entry is set.

The privilege flag in the user directory entry associated with each task to be executed is transferred to the descriptor block and alternator tables. This provides the operating system with a means of ensuring that the only users who issue privileged messages and message options are privileged users.

TABLE 1-1. RESIDENT SYSTEM CALLS FOR PRIVILEGED USERS

Call Number	Meaning	Call Number	Meaning
F002	Delete pages from the page table.	C300	Read system service station storage.
F003	Delete a virtual range with a given key from the page table.	C301	Write system service station storage.
F004	List a virtual range with a given key from the page table.	C304	Teletype output message.
F005	Delete all pages in the page table under a given key.	C305	Teletype input message.
F007	List the page table entry for the keyword and the virtual page identifier.	C30E	Initiate USER1 routine from the station.
F008	Terminate the task.	C311	Service station queue/dequeue.
F00C	Change key.	C312	Service station drum file function.
F00D	Get an input buffer.	C313	Full screen output message.
F00E	Queue system/demand task.	C314	Service station block transfer function.
F00F	Change system keys.	C320	Reserved for installation use.
F010	Unlock a virtual range with a given key.	C400	Load a tape.
F011	Link I/O parameter call.	C401	Unload a tape.
F012	Read a link file.	C402	Read a tape record.
F013	Write a link file.	C403	Write a tape record.
F014	Write an EOF for a link file.	C404	Tape function.
F015	Performance measurement call.	C500	Read a page.
F016	Process checkpoint.	C501	Write a page.
F017	Return to KERNEL from VSDT.	C502	Read the disk.
FF00	Advise.	C503	Write the disk.
FF01	Process system parameter.	C504	Write disk pattern.

Privileged Resident System Calls

Privileged tasks (as well as the virtual system) can make calls to KERNEL. The calls are processed by KERNEL or by a station. Resident system calls available to privileged tasks are listed in table 1-1.

Before a C501, C502, or C503 call on a file can be issued by a privileged task, the file must be opened for explicit input/output, using the OPEN FILE (f=0003) message. The file segment table (FST) ordinal (returned in the fsto Beta word field for a CREATE or OPEN message) must be supplied with the call; otherwise, the task aborts.

Privileged tasks can read and write segmented files with C500 and C501 calls, but files are not extendable by tasks using these calls. Extensions are permitted, however, in other situations such as when the privileged tasks perform implicit input/output or use the EXPLICIT I/O (f=0050) message.

Virtual System Calls

Privileged tasks can make all nonprivileged calls to the virtual system. They can also make special virtual system calls and can use options of nonprivileged calls that are restricted to use by privileged tasks only. (All messages described in section 4, except ADVISE and PROCESS SYSTEM PARAMETER, are virtual system calls.)

Special virtual system calls available to privileged tasks are listed in table 1-2. The restricted capabilities of nonprivileged calls available to privileged tasks are listed in table 1-3.

TABLE 1-2. VIRTUAL SYSTEM CALLS AVAILABLE TO PRIVILEGED USER TASKS ONLY

Function Code	Message	Function
0011	GET PACK LABEL AND PFI	This call is made by a privileged user to retrieve the pack label and pack file index for a specified pack.
0021	EXECUTE OPERATOR COMMAND	This call is the interface between the privileged task OPERATOR and the virtual system.
0022	EXECUTE PROGRAM FOR USER NUMBER	This call is made by a privileged task to start the batch processor for a user.
0023	UPDATE USER DIRECTORY	This call is made by a privileged task to modify the user directory of an existing user or to create a new user.
0027	LINK	Reserved for operations relating to the CYBER 200 Link Station.
0030	EXECUTE IQM REQUEST	This call is made by a privileged task to start the input queue manager.

TABLE 1-3. VIRTUAL SYSTEM CALLS WITH OPTIONS AVAILABLE TO PRIVILEGED USER TASKS ONLY

Function Code	Message	Option
0001	CREATE	Create a public file, pool file, or private file for another user.
0002	DESTROY	Destroy a public file, pool file, or another user's private file.
0003	OPEN	Open any file: private, public, or pool.
0005	CLOSE	Close any public, private, or pool file. Destroy the file being closed.
0008	GIVE	Make a file public; that is, give to user 000000. Give file to IQM. Give file specified by sector address and unit number.
000B	CHANGE FILE ATTRIBUTES	Change the account number of an input file (used by CARDREDR).
000E	USER/ACCOUNTING COMMUNICATION	Start or end of the batch job. Dump accounting temporary storage to permanent storage and terminate the accounting file.

PERIPHERAL OPERATING SYSTEM

The peripheral system for all station processors has two parts:

- A resident system called NUCLEUS, a copy of which resides in each station.
- A nonresident set of overlays for performing tasks related to the responsibilities of the individual stations.

Each station uses its set of nonresident tasks to control peripheral equipment. Operating system tasks for each station are stored on its microdrum.

NUCLEUS consists of diagnostic routines, a station deadstart program, driver programs for the microdrum and keyboard/display, an organizational program, programs to manage the system overlay mechanism, and SCANNER, which is the main control and organizational program. SCANNER uses scan bits and an associated table to determine which station routines to call. If a particular routine to be run is not in the station's main memory, an overlay driver in NUCLEUS calls in the routine from the station's microdrum.

Nonresident tasks are concentrated into larger processing routines to facilitate on-line error handling and maintenance procedures common to all stations.

Tables used by CYBER 200 OS to control job processing within the system can be affected or altered indirectly by user programs. In all cases, access to the tables must be through system messages.

FILE INDEX TABLE

The file index table is a catalog of magnetic tape and mass storage files (public, private, and attached pool) for all active users in the system. The catalog entries of mass storage files for inactive users are maintained on a mass storage unit. When a user becomes active, the catalog entries describing the user's mass storage files are brought into the file index table in main memory. Each entry in the file index table consists of 16 words in the format shown in figure 2-1. All fields in an entry contain binary values unless indicated otherwise.

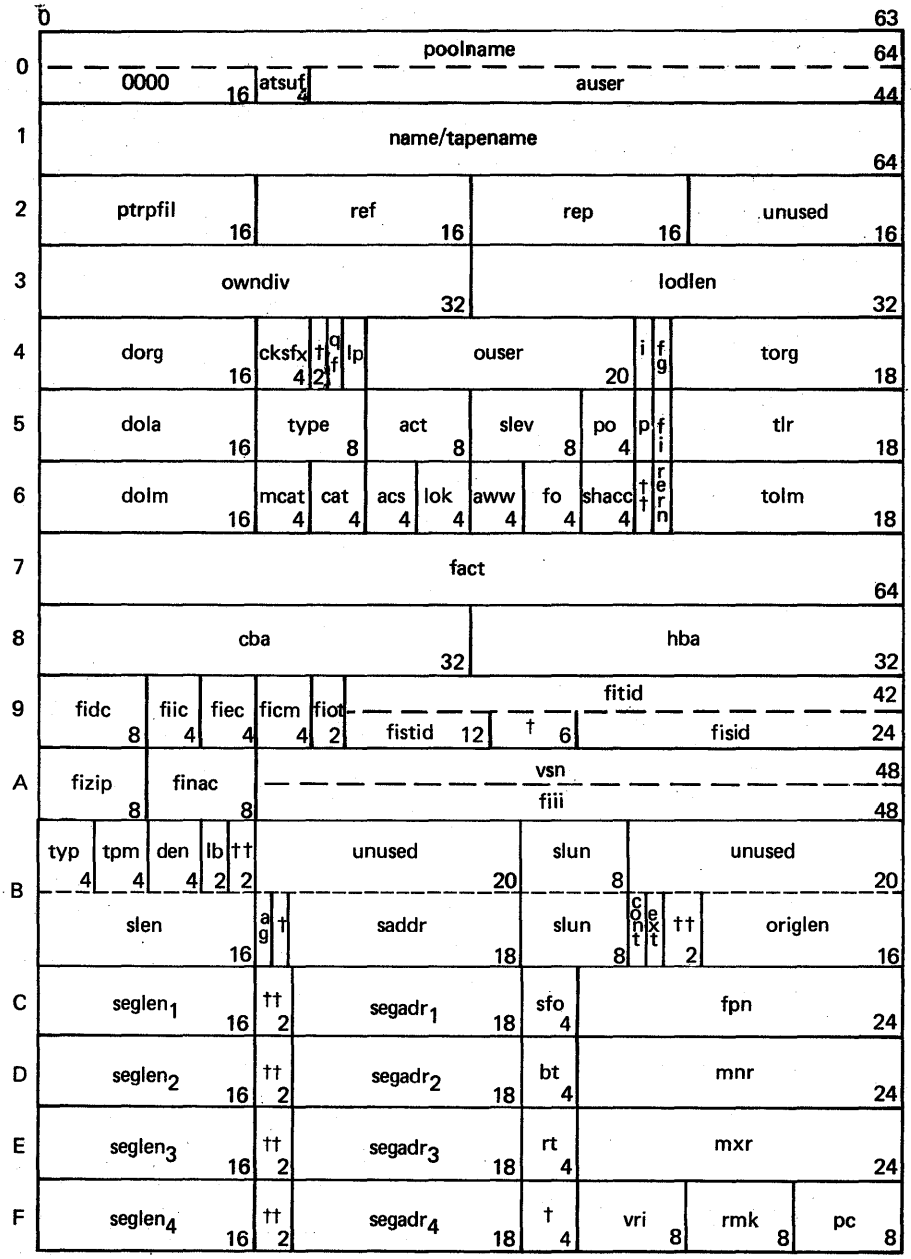
The fidc, fiic, and fiec fields in the file index table entry contain specifications for the file's disposition, internal characteristics, and external characteristics. The fidc field values are listed in table 2-1; the values of the fiec and fiic fields are listed in table 2-2.

TABLE 2-1. FILE DISPOSITION SPECIFICATIONS IN FILE INDEX TABLE

Destination	Value	Mnemonic	Disposition
Any	0	None	No disposition.
	1	SC	Scratch.
	#10	PU	Punch.
	#20	PR	Any available line printer.
Access Station	8	IN	Input for batch processing.
	9	PF	Permanent file.
CYBER 200 Link Station	#21 #22 #23 #24 #25	P1 P2 LR LS LT	Reserved.

TABLE 2-2. FILE CHARACTERISTICS SPECIFICATIONS IN FILE INDEX TABLE

Type	Value	Mnemonic	Format
Internal characteristics (fiic field in word 9)	0	none	Default PA.
	1	PA	8-bit ASCII. If the fidc field indicates a print file, the file has free-form carriage control.
	2	BI	Binary.
	3	AS	8-bit ASCII. If the fidc field indicates a print file, the file has ANSI-defined carriage control.
External characteristics for punch files (fiec field in word 9)	0	none	Default 29.
	1	29	O29 keypunch.
	2	26	O26 keypunch.
	3	80	80-column binary.
External characteristics for print files destined for the CYBER 200 Link Station only (fiec field in word 9)	5	A4	Reserved.
	6	B4	
	7	B6	
	8	A6	
	9	A9	



†Unused. ††Reserved.

Word

0 poolname Pool name in ASCII, left-justified with blank fill.

atsuf Attached suffix:

0	File is unattached
≠0	File is attached to the suffix given (1, 2, 3, or 4 corresponding to suffix A, B, C or D)

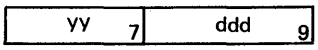
auser Owning user number.

1 name File name in ASCII, left-justified with blank fill. The information in this table refers to the file of this name. File names must be in the format described in section 3.

tapename File name used to assign the tapes.

Figure 2-1. File Index Table Format (Sheet 1 of 5)

Word		
2	ptrpfil	Pointer into the proper page of the pack file index (PFI) for this entry relative to the first page of the PFI. PFI = #FFFF for local files.
	ref	Number of times this file has been opened.
	rep	Retention period, in days.
3	owndiv	Division code of the file owner in ASCII, right-justified with blank fill.
	lodlen	Drop file length (in small pages) for the execute file; only valid if the file is a virtual code file (the type field is 2). For a file given to a USER1 routine, this field contains the binary user number of the user who gave this file to the USER1 routine.
4	dorg	Date this file was originated, in the format:



yy	Last two digits of the year
ddd	Number of days since the beginning of the year (1 through 366)

cksfx — Checkpoint suffix. Set when this file belongs to a checkpointed job.

0	Does not belong to a checkpointed job
1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

lp — Local/permanent flag; 0=perm, 1=local.

qf — Queue flag.

0	Input queue manager has read the file.
1	Input queue manager has not read the file.

ouser — Originating user number.

i — Duplicate files flag:

0	No duplicate file was found
1	Duplicate file was found at logon

A duplicate file is a pre-existing file having the same name and same ownership as file for this file index table entry.

fg — File acquisition method:

0	User created this file
1	File was given to the user

torg — System clock time, in seconds since midnight, at which this file was originated.

5	dola	Date of the last access to this file, in the same format as the dorg field.
	type	File type:

0	Physical file
1	Virtual data file
2	Virtual code file

Figure 2-1. File Index Table Format (Sheet 2 of 5)

Word		
	act	Number of active input/output connectors for this file.
	slev	Security classification level of this file (0 through 255).
	po	Count of privileged opens of this file that are currently active.
	p	Privileged open designator: 0 Regular open 1 Privileged open
	fi	Task privilege designator: 0 Not a privileged task 1 Privileged task
	tlr	System clock time, in seconds since midnight, at which the file was last opened.
6	dolm	Date the file was last opened with write access, in the same format as the dorg field.
	mcat	Present file management category: 0 Mass storage file 1 Scratch file 2 Output file 3 Write-temporary file 4 Magnetic tape file 5 User-generated drop file 6 System-generated drop file 7 Batch file
	cat	File management category for the first open of this file: 0 Mass storage file 1 Scratch file 2 Output file 3 Write-temporary file 4 Magnetic tape file 5 User-generated drop file 6 System-generated drop file 7 Batch file
	acs	Default file access permission: 0 No access is allowed (same as acs=4) 1 Write access is allowed (same as acs=5) 2 Read access is allowed (same as acs=6) 3 Read and write access are both allowed (same as acs=7)
	lok	File access lockout: 0 No access is locked out (same as lok=4) 1 Write access is locked out (same as lok=5) 2 Read access is locked out (same as lok=6) 3 Read and write access are both locked out (same as lok=7)
	aww	Count of active users with write access to this file.
	fo	File organization: 0 Unstructured binary 1 Structured 2 Unstructured ASCII
	shacc	Shared access permission for pool and public file sets: 0 No shared access is permitted (same as shacc=4) 1 Write shared access is permitted (same as shacc=5) 2 Read shared access is permitted (same as shacc=6) 3 Read and write shared access are both permitted (same as shacc=7)

Figure 2-1. File Index Table Format (Sheet 3 of 5)

Word		
	rern	Rerun indicator, valid for batch input files (the mcat field is 7) only: 0 Batch input file is to be rerun if any batch job in the file is unable to complete because of a system failure 1 Batch input file is to be destroyed if any batch job in the file is unable to complete because of a system failure
	tolm	Time, in seconds since midnight, when this file was opened with write access.
7	fact	Account number of the file in ASCII, left-justified with blank fill.
8	cba	Reserved for the byte address of the current byte being accessed.
	hba	The byte address of the highest page accessed plus one byte; zero if the file was created before the hba field was implemented.
9	fidc	Numeric value, indicating disposition of this file (table 2-1).
	fiic	Numeric value, indicating the internal format characteristics of the file (table 2-2).
	fiec	Numeric value, indicating the external punch representation characteristics of the file (see table 2-2).
	ficm	Numeric value, indicating the conversion mode of the file. This field is meaningful only if the file originated from, or is destined for, the access station. The values are: 0 Display code (64-character set) 1 Extended display code (128-character set) 2 Binary
	fiot	Numeric value, indicating the origin type of the file. This field is meaningful only if the file originated from, or is destined for, the access station. The values are: 0 Local batch 1 Remote batch 2 Interactive
	fitid	A seven-character terminal identifier from the access station, stored as seven 6-bit display code characters.
	fistid	A two-character terminal identifier from the CYBER 200 Link Station, stored as two, 6-bit display-code characters.
	fisid	For the CYBER 200 Link Station, a three-character (ASCII) mainframe identifier identifying the source or destination mainframe.
A	fizip	Numeric value indicating the destination processor zip number for this file. The mainframe table associates this zip with a three-character mainframe identifier.
	finac	Access station area code.
	vsn	Volume serial number if magnetic tape file. This is a 6-character identifier for tape, displayed as part of the load request.
	fiii	For the access station, eight characters of delivery information. It is stored as eight 6-bit display code characters.
B	typ	Device type if magnetic tape file. 7 7-track tape 9 9-track tape
	tpm	Tape mode if magnetic tape file. 0 External BCD (7-track) 1 Unformatted binary/8-bit ASCII (7- or 9-track) 2 6-bit ASCII (7-track)

Figure 2-1. File Index Table Format (Sheet 4 of 5)

Word		
	den	Density if magnetic tape file. 0 200 bpi 1 556 bpi 2 800 bpi 3 1600 bpi
	lb	Tape label format if magnetic tape file. 0 Standard ANSI labels 1 Unlabeled
	slen	Length of the file in small pages.
	ag	Active segment flags used in the pack file index for determining whether this segment is on this unit: 0 Segment is not active 1 Active segment on this unit
	saddr	Starting block address on disk for the first segment (same as $segadr_1$).
	slun	Logical unit number of this segment.
	cont	File contiguity, set at file creation time: 0 Not contiguous (in two segments) 1 Contiguous
	ext	File extendability, set at the time the file was created or opened: 0 Extensions allowed 1 No extensions allowed
	origlen	Original file length (in small pages) requested during file creation.
C	sfo	File organization. 0 Sequential 1-7 Unused
	fpn	File position number if magnetic tape file. A sequential file number currently being processed on a multi-file tape.
D	bt	Blocking type. 0 Non-SIL file 2 C type blocking
	mnr	Minimum record length; 24-bit user-supplied length in number of bytes.
E	rt	Record type. 0 Control word (W) 1 ANSI fixed length (F) 2 Record mark (R) 7 Undefined (U)
	mxr	Maximum record length; 24-bit user-supplied maximum length in number of bytes
F	vri	Variable rate index transferred to the db to be used in MCHARGE for variable rate accounting. Valid for virtual code files only.
	rmk	Record mark; 8-bit ASCII character (any character is valid).
	pc	Padding character; 8-bit ASCII character (any character is valid).
C, D, E, F	$seglen_i$	Length of the i^{th} segment of the file in small pages. If $seglen_i$ is zero, the file contains only i-1 segments.
	$segardr_i$	Starting block address on disk of the i^{th} segment of this file.

Figure 2-1. File Index Table Format (Sheet 5 of 5)

MINUS PAGE

CYBER 200 OS recognizes two types of files: virtual files, usually containing executable code (although some can contain data), and physical files, always containing nonexecutable data only. A virtual file is prefaced by a 512-word block containing program execution and data access information to be used by the operating system. This preface is known as the minus page. It is created either by the operating system at load time or by the user. Physical files do not require minus pages.

The minus page of a virtual code file is created at load time to pass information such as the entry point address, the length of the drop file, the code origin, data base locations, and so forth. The operating system needs this information for starting program execution. Once execution begins, the operating system uses the minus page to store invisible packages, time-slicing data, input/output connection blocks to high-speed devices, maps of defined virtual space, time-sharing data, and statistics relating to resource use. If the program execution terminates abnormally, the minus page is stored on the drop file and can be used for debugging purposes. The original minus page remains on the virtual code file as initialized at load time. In addition, drop files contain a second minus page (immediately following the first), which is a logical extension of the first minus page.

Virtual data files have a minus page that might contain virtual-to-physical address mapping information. This information must be set by the user.

An executing controllee cannot access its minus page except via system messages. A controllee can execute the SIL call Q5GETMPG to copy an interrupted task's minus page. Otherwise, the minus page is like any other part of a file, and can be accessed explicitly or implicitly.

The minus page has the format shown in table 2-3. Individual words are described in the table, with the contents of the invisible package shown in appendix E.

The working set pager information buffer in the minus page is used by PAGER to store information about a task's working set. The buffer occupies words #28 through #31.

INPUT/OUTPUT CONNECTORS

Words #40 through #87 of the first minus page contain the first 18 input/output connectors. Words #2 through #D9 of the second minus page contain the remaining 54 input/output connectors. An input/output connector (ioc) is a four-word block used to establish a link between the program and an input/output device. The operating system also uses input/output connectors to keep track of the activity of a specific file and a program's use of that file. Each time a program issues a CREATE FILE or OPEN FILE request, an input/output connector is created and initialized by the system with information provided in the request and in the file index table.

Each program can have up to 70 connectors for user files, numbered 0 through #F and #12 through #47. The input/output connector for the program's source file is numbered #10, and the input/output connector for the program's drop file is #11. Input/output connectors numbered 0 through #F and #12 through #47 can be allocated by the user or automatically allocated by the system.

Formats of the input/output connectors are illustrated in figures 2-2, 2-3, and 2-4. The connector for a mass storage file opened for explicit input and output is shown in figure 2-2; the connector for a mass storage file opened for implicit input and output is shown in figure 2-3; and the connector for a magnetic tape file opened for explicit input/output is shown in figure 2-4. (A magnetic tape file cannot be opened for implicit input and output.)

In figure 2-3, when the name field contains a drop file name, the fourth word of the input/output connector serves the same purpose as does the second word of a bound implicit map entry for a source file.

EXPLICIT INPUT/OUTPUT INFORMATION

Word #97 of the first minus page contains information that, along with the interrupt address stack, is required by the operating system for processing explicit input and output messages. It is formatted as shown in figure 2-6.

The leftmost 16 bits of word 88 of the first minus page contains the second minus page pointer. If this field contains a zero or #FFFF, there is no second minus page. Otherwise, this field contains the physical page address of the second minus page.

MAP DIRECTORIES

Words #88 through #8A of the first minus page contain map directories. Each map directory contains information relating to the location and length of its associated file map. Each directory occupies the second half-word of its location in the minus page. For the first minus page, the bound explicit map directory is at word #88; the bound implicit map directory is at word #89; and the drop file map directory is at word #8A. Each directory is formatted as shown in figure 2-7. For the second minus page, the bound explicit map directory is at word 0, and the bound implicit map directory is at word 1.

TABLE 2-3. MINUS PAGE FORMAT

Words (base 10)	Words (base 16)	Contents
0 to 39	0 to 27	Executing program invisible package.
40	28	Unused.
41 to 49	29 to 31	Working set pager information buffer.
50 to 52	32 to 34	Program restart temporary buffers.
53, 54	35, 36	Time information required by the operating system for alternator and message management.
55, 56	37, 38	Same as words #32 to #34.
57, 58	39, 3A	Unused.
59 (bits 0 to 15)	3B	Error code saved during abnormal termination control processing.
59 (bits 16 to 63)	3B	Pointer to SIL FNT and FIT common block.
60, 61	3C, 3D	Abnormal termination control interrupt subroutine and status information.
62, 63	3E, 3F	Interrupt data base pointers.
64 to 127	40 to 7F (2-D9)	Input/output connectors for user disk or tape files.
128 to 131	80 to 83	Input/output connector for the source file.
132 to 135	84 to 87	Input/output connector for the drop file.
136 (bits 0 to 15)	88	Second minus page pointer.
136 (bits 16 to 23)	88	Used by set indefinite.
136 (bits 24 to 31)	88	Unused.
136 (bits 32 to 63)	88	Directory for bound explicit map entries.
137 (bits 0 to 31)	89	Unused.
137 (bits 32 to 63)	89	Directory for bound implicit map entries.
138 (bits 0 to 31)	8A	Unused.
138 (bits 32 to 63)	8A	Directory for drop file map entries.
139	8B	Error number (appendix B) of a fatal error condition and the address where it occurred.
140 to 150	8C to 96	Time usage and accounting entries.
151	97	Explicit input/output information.
152 to 159	98 to 9F	Interrupt address stack for input/output and terminal interrupts.
160 to 175	A0 to AF (DA-10F)	Bound explicit maps (of files opened for explicit input or output).
176 to 255	B0 to FF (110-1FF)	Bound implicit maps (of files opened for implicit input or output).
256 to 511	100 to 1FF	Drop file map.

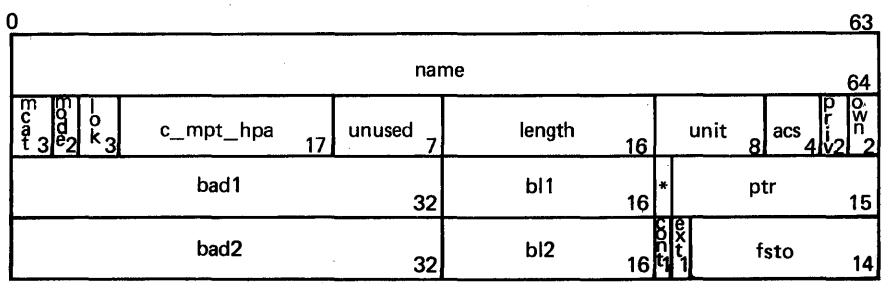
Words shown in parentheses are second minus page values.

INTERRUPT ADDRESS STACK

The operating system uses words #98 through #9F of the first minus page (along with word #97) for processing interrupts from explicit input/output messages and from terminals. With eight requests for input/output possible for each of six possible explicit input/output messages, multiple requests for input/output are likely to occur.

If an interrupt routine is already in control when additional interrupt conditions are met, the interrupt information is stacked in the interrupt address stack by the virtual system. The interrupts so stacked are processed until the stack is empty before control finally returns to the executing program.

The six words of the stack correspond to the six 8-bit input/output fields in word #97. Each of the six words contains control fields and a pointer to the Alpha word for the input/output request. The stack is formatted as shown in figure 2-7.



name — Name of the file in ASCII. File names must be in the format described in section 3.

mcat — File management category:

- 0 Mass storage file
- 1 Scratch file
- 2 Output file
- 3 Write-temporary file
- 4 Magnetic tape file
- 5 User-generated drop file
- 6 System-generated drop file
- 7 Batch file

mode — Mode of input/output:

- 0 Open for explicit input and output
- 1 Open for implicit input and output

lok — File access lockout:

- 0 No access is locked out (same as lok=4)
- 1 Write access is locked out (same as lok=5)
- 2 Read access is locked out (same as lok=6)
- 3 Read and write access are both locked out (same as lok=7)

c_mpt_hpa — Highest page accessed. Zero if no pages have been accessed. Set to the length of the file if the file was created before the hba field in the file index was implemented.

length — Length of the file in small page blocks.

unit — Logical unit number.

acs — Default file access permission:

- 0 No access is permitted (same as acs=4)
- 1 Write access is permitted (same as acs=5)
- 2 Read access is permitted (same as acs=6)
- 3 Read and write access are both permitted (same as acs=7)

priv — Privileged open designator:

- 0 Regular open
- 1 Privileged open

own — File ownership (see section 3). The values are:

- 0 Private
- 1 Public
- 2 Pool

bad1 — Virtual page address of the first input/output buffer.

bl1 — Length in small pages of the first input/output buffer.

ptr — Owner's user table (UT) entry or pool list (PLIST) table entry number for a privileged open file.

bad2 — Virtual page address of the second input/output buffer.

*f_mt_rhba — Flag to indicate reset hba at close time.

- 0 Do not reset hba
- 1 Reset hba

bl2 — Length in small pages of the second input/output buffer.

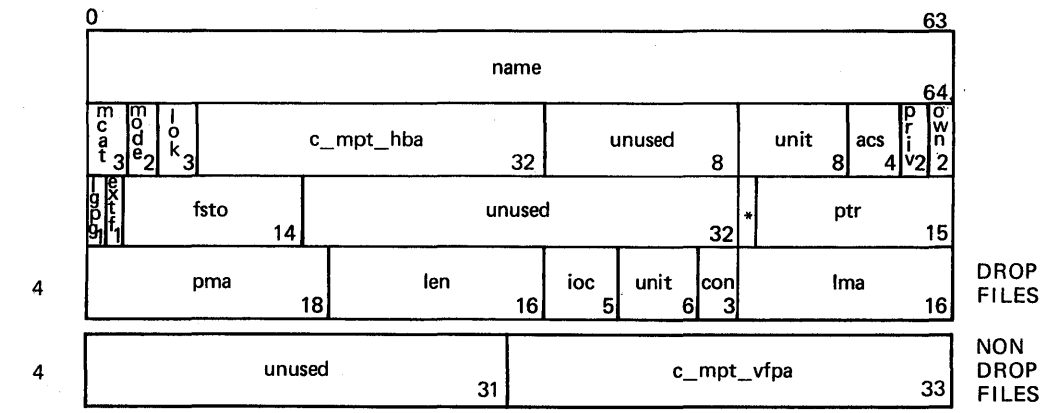
cont — File contiguity, specified at file creation time.

- 0 Not contiguous (in two segments)
- 1 Contiguous

Figure 2-2. Format of Input/Output Connector for a Mass Storage File Opened for Explicit Input/Output (Sheet 1 of 2)

ext	File extendability, specified at the time the file was created or opened: 0 Extensions allowed 1 No extensions allowed
fsto	File segment table (FST) ordinal.

Figure 2-2. Format of Input/Output Connector for a Mass Storage File Opened for Explicit Input/Output (Sheet 2 of 2)



name — Name of the file in ASCII. File names must be in the format described in section 3.

mcat — File management category:

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
4	Magnetic tape file
5	User-generated drop file
6	System-generated drop file
7	Batch file

mode — Mode of input/output:

0	Open for explicit input and output
1	Open for implicit input and output

lok — File access lockout:

0	No access is locked out (same as lok=4)
1	Write access is locked out (same as lok=5)
2	Read access is locked out (same as lok=6)
3	Read and write access are both locked out (same as lok=7)

c_mpt_hpa — Highest page accessed. Zero if no pages have been accessed. Set to the length of the file if the file was created before the hba field in the file index was implemented.

unit — Logical unit number.

acs — Default file access permission:

0	No access is permitted (same as acs=4)
1	Write access is permitted (same as acs=5)
2	Read access is permitted (same as acs=6)
3	Read and write access are both permitted (same as acs=7)

priv — Privileged open designator:

0	Regular open
1	Privileged open

own — File ownership (see section 3). The values are:

0	Private
1	Public
2	Pool

lgpg — Extensions page size; this field has meaning only when the extf field is set. The values are:

0	Small pages
1	Large pages

Figure 2-3. Format of Input/Output Connector for Implicit Input and Output (Sheet 1 of 2)

extf Set to 1 by the operating system if the file needs to be extended.

fsto File segment table (FST) ordinal.

*f_mpt_rhba Flag to indicate reset hba at close time.

0 Do not reset hba
1 Reset hba

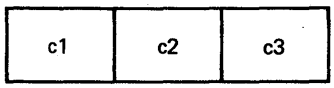
ptr Owner's user table entry or pool list (PLIST) table entry number for a privileged open file.

pma For a drop file, the physical mass storage sector address of this file's first page; otherwise, zero.

len For a drop file, the length of the file in small page blocks; otherwise, zero.

ioc For a drop file, the file's input/output connector number; otherwise, zero.

con For a drop file, a control field of the following format:



c1=1 Write access is permitted
c2=1 Read access is permitted
c3=0 File is contained on small pages
c3=1 File is contained on large pages

Otherwise, zero.

lma For a drop file, the logical mass storage sector address of this file's first page; otherwise, zero.

c_mpt_vfpa Virtual page address of the start of the file. Zero if the file is not mapped in from the start of the file.

Figure 2-3. Format of Input/Output Connector for Implicit Input and Output (Sheet 2 of 2)

MINUS PAGE FILE MAPS

The file maps in the minus pages relate files to physical mass storage areas. For files opened for implicit input and output, the maps relate physical mass storage areas to virtual address areas. Each time a program opens a file for explicit input or output, one entry is made in a bound explicit map. Each time a program opens a file for implicit input or output, entries may or may not be made in a bound implicit map. The MAP message (f=0004) places entries in a bound implicit map.

Bound Explicit Maps

Words #A0 through #AF of the first minus page and #DA through #10F in the second minus page contain bound explicit maps. These maps are related to files opened for explicit input and output (mode=0). Each file that has been opened for explicit input and output corresponds to one map entry; the files are identified by their input/output connector numbers. The format of the bound explicit map entry is shown in figure 2-8.

Bound Implicit Maps

Words #B0 through #FF of the first minus page and #110 through #1FF in the second minus page contain bound implicit maps. These maps are related to files opened for implicit input and output (mode=1); such files can consist of discontinuous virtual address ranges. Up to 160 virtual address space segments can be mapped simultaneously. All the segments can be associated with one input/output connector, or each segment can be so associated. The format of a bound implicit map entry is shown in figure 2-9.

In bound implicit map entries, all first words are in the first half of the map space, and all second words are in the second half. Entries are sorted by ascending virtual page address; blank entries are squeezed out. Note that both minus pages have a 'first half of the map space' so that no map entry is split between the two minus pages.

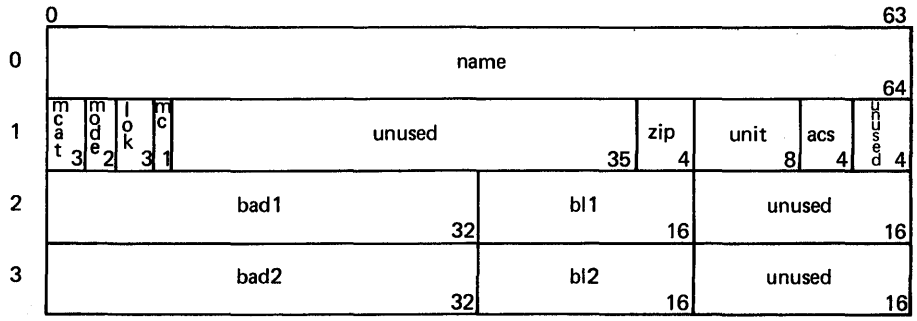
DROP FILE MAP

Half of the minus page is a drop file map. In the case of a free-space attachment to a file, the defined space is allocated a part of the drop file on which it can reside if it becomes necessary to swap the attachment out. Free-space attachments are cataloged in the drop file map in much the same way that other kinds of virtual space are cataloged in the bound implicit map.

Each drop file map entry consists of one full word and one half word. Up to 170 entries can be made in the drop file map, and each entry can have up to 31 associated pages. This allows for up to 170 noncontiguous address spaces to be part of the drop file.

The 170 full-word entries occur first in the map space. The format of these entries is shown in figure 2-10. The 170 half-word entries that follow the full-word entries correspond as shown in figure 2-11.

Each half-word entry consists of 31 bits, one bit per page. If the bit is zero, the page is either undefined or exists in main memory or on the paging device; if the bit is 1, the page has been written to mass storage on the drop file. Bit 0 or 32 corresponds to page 1 of a segment; bit 30 or 62 corresponds to page 31 of a segment.



name — Name of the file in ASCII. File names must be in the format described in section 3.

mcat — File management category:

0	Mass storage file	4	Magnetic tape file
1	Scratch file	5	User-generated drop file
2	Output file	6	System-generated drop file
3	Write-temporary file	7	Batch file

mode — Mode of input and output:

0	Open for explicit input and output
1	Open for implicit input and output

lok — File access lockout:

0	No access is locked out (same as lok=4)
1	Write access is locked out (same as lok=5)
2	Read access is locked out (same as lok=6)
3	Read and write access are both locked out (same as lok=7)

mc — Reserved for specifying that this input/output connector is for a CYBER 200 Link Station file and not a tape file.

zip — Reserved for CYBER 200 Link Station file's station zip.

unit — Physical unit number, for tape files. Reserved for subchannel number, for CYBER 200 Link Station files.

acs — Default file access permission:

0	No access is permitted (same as acs=4)
1	Write access is permitted (same as acs=5)
2	Read access is permitted (same as acs=6)
3	Read and write access are both permitted (same as acs=7)

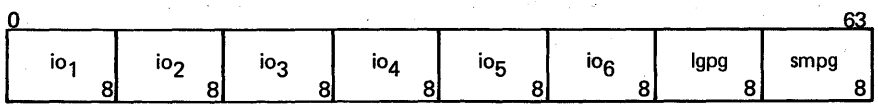
bad1 — Virtual page address of the first input/output buffer.

bl1 — Length in small pages of the first input/output buffer.

bad2 — Virtual page address of the second input/output buffer.

bl2 — Length in small pages of the second input/output buffer.

Figure 2-4. Format of Input/Output Connector for Tape and Link Files



io_i — A set of eight 1-bit flags used to keep track of the number of outstanding explicit input/output requests (six explicit input/output calls can be outstanding at any time for a particular program, one request for each io field). An EXPLICIT INPUT OR OUTPUT message (f=0050) can stack up to eight Beta requests; an outstanding Beta request is indicated by a bit set in the io field.

lgpg — Number of large pages for which input or output requests are outstanding.

smpg — Number of small pages for which input or output requests are outstanding.

Figure 2-5. Format of Word #97 of the Minus Page

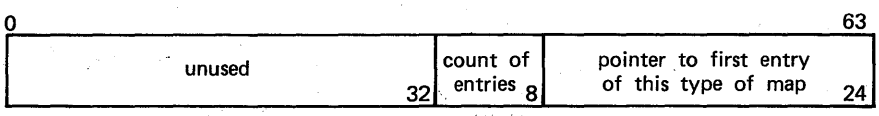
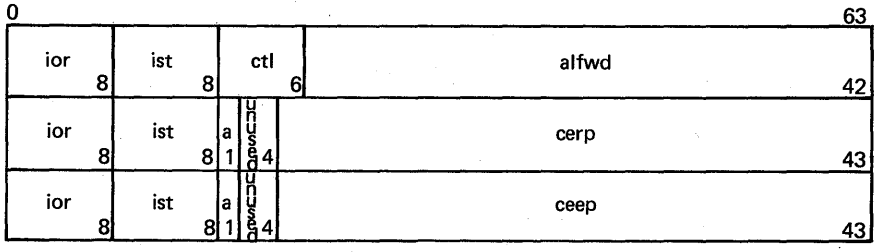


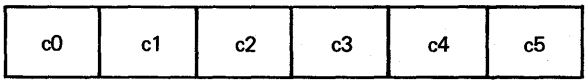
Figure 2-6. Map Directory Format



ior — Last input/output Beta request processed or in process, an index (1 to 8) into the Beta requests stacked in the message indicated by alfwd.

ist — One bit for each of the eight potentially outstanding Beta requests, or one bit for the controller or controllee of this program. A set bit indicates an interrupt condition has been met and is stacked.

ctl — Flag bits, in the following format:



c0=1 GIVE UP CPU UNTIL I/O COMPLETES message has been issued for this request by the main level task.
c1=1 Unused
c2=1 GIVE UP CPU UNTIL I/O COMPLETES message has been issued for this request by the interrupt level task
c3 Unused
c4=1 Program is currently the interrupt level task for this request
c5 Unused

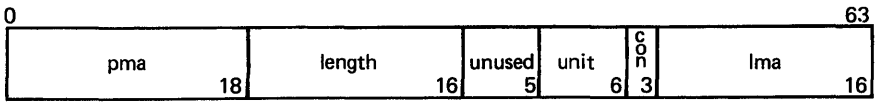
alfwd — Alpha word pointer to the input/output request; if there is an interrupt address, it is in the first Beta word.

a — If this bit is set, only messages preceded by the characters (sc)I will interrupt (sc is an installation-defined special character).

cerp — Controller interrupt program address counter (half-word virtual address).

ceep — Controllee interrupt program address counter (half-word virtual address).

Figure 2-7. Interrupt Address Stack Format

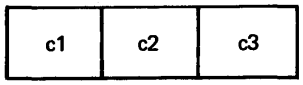


pma — Physical mass storage sector address of this file's first page.

length — Length of the file in small pages.

unit — Logical unit number.

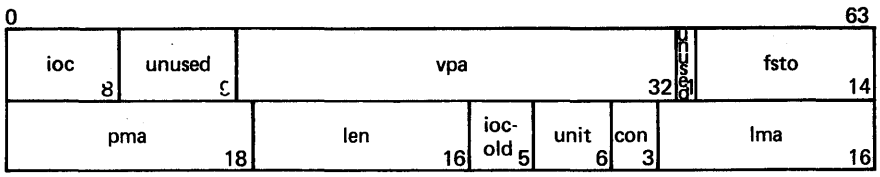
con — Flag bits, in the following format:



c1=1 Write access is permitted
c2=1 Read access is permitted
c3=0 File consists of small pages
c3=1 File consists of large pages

lma — Logical mass storage sector address of this file's first page.

Figure 2-8. Bound Explicit Map Entry Format



ioc — Input/output connector number associated with this segment.

vpa — Virtual page address of the first page of this segment.

fsto — File segment table (FST) ordinal.

pma — Physical mass storage sector address of this segment's first page.

len — Length of this segment in small pages.

iocold — Reserved for release 1.2/1.3 compatibility.

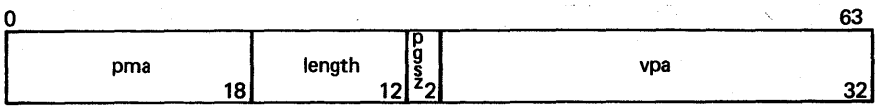
unit — Logical unit number.

con — Control field. (See con under Bound Explicit Maps. See figure 2-9.) The values are:

c1 Unused
c2 Unused
c3=0 Segment consists of small pages
c3=1 Segment consists of large pages

lma — Logical mass storage sector address of the first page of this segment.

Figure 2-9. Bound Implicit Map Entry Format



pma — Physical mass storage sector address of the first page of this segment of virtual space.

length — Length of this segment of virtual space in small pages.

pgsz — Size of the pages in this segment of virtual space:

0 Small pages
1 Large pages

vpa — Virtual page address of the first page in this segment of virtual space.

Figure 2-10. Drop File Map Entry Format

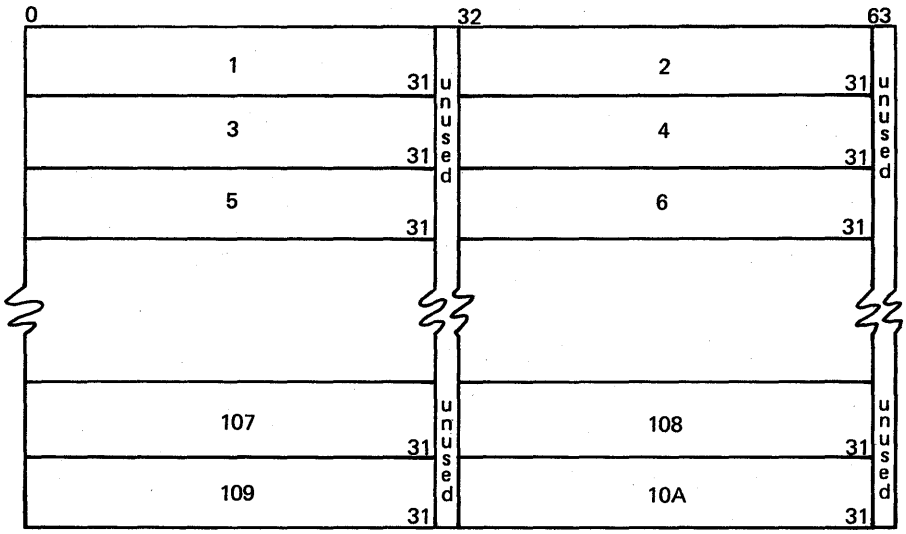


Figure 2-11. Drop File Map Format

The file concepts used in the CYBER 200 OS are described in volume 1. This section describes additional file specifications used in the system messages described in section 4.

FILE NAMES

A file has only one name associated with it; it is both its permanent file name and its logical file name. The file name must be up to eight letters and digits long, left-justified and blank-filled within the field. File names cannot contain special characters. User-created file names must begin with a letter. System-created file names can begin with a leter or a digit. The conventions in naming drop files are described in volume 1.

FILE OWNERSHIP

Privileged users have ownership rights over all files except local files. A nonprivileged user has ownership rights determined by the values in file index table fields as shown in table 3-1. The file index table is described in section 2. The GIVE FILE system message can change file ownership.

The file owner specifies the file attributes. To permanently change file attributes, the appropriate fields in the file index table must be changed (refer to the CHANGE FILE ATTRIBUTES system message in section 4).

The three file ownership categories are private, pool, and public. Each private file cataloged in the file index table belongs to a particular user number and account identifier. When a private file is given by one user to another, the user number associated with the file changes immediately. However, the account identifier does not change until the new owner references the file. The system accounting tables then indicate the total amount of time that the original account owned the file.

Calls to the POOL FILE MANAGER system message perform the same functions as the pool file utilities described in volume 1. The GIVE system message can give files to a pool; the DESTROY FILE system message can destroy pool files.

Public files are owned by user number 000000, signifying system ownership. The list of public files is controlled by the installation administrator or by privileged tasks.

FILE ACCESS AND LOCKOUT

File access is controlled by the file ownership category, file access permission and lockout fields in the file index table entry. All users can access public files; users given access to a pool can access files in the pool; the file owner can access private files. (The exception to this rule is the privileged user who can access any permanent file in the system.) Any virtual code file a user can access can also be executed by that user.

Read and write access are controlled by the permission and lockout parameters, which are set when the file is created. A file with write access can be written into by a user program or by the operating system. For data files, this means the system returns modified pages to the original file, rather than to the drop file.

An attempt to write explicitly into a read-only file produces an input/output error. An attempt to modify a page from a read-only virtual file produces a fatal error. However, the MAP message (f=0004) can map in pages of read-only files and give those pages write-temporary access.

When the user opens a file, he requests read-only, write-only, or read and write access. The system checks if the requested access is allowed for the file by checking the access lockout bits. For example, if write lockout has been specified for the file, write access cannot be granted to the file.

TABLE 3-1. FILE INDEX TABLE FIELDS THAT AFFECT FILE OWNERSHIP

File Category	File Index Table Field	Nonprivileged Owner	Users Allowed Access
Public	Owndiv all 1's	All users	All users
	Owndiv all 0's Ouser not 0	Originating user†	
	Owndiv not all 0's and not all 1's	None	
Pool (poolname=0)	None	Pool boss	Users according to pool access list of user numbers
Private	None	Originating user†	Originating user†

†Originating user number determined by the user field of the file index table.

FILE MANAGEMENT CATEGORIES

The management category field in the file index table entry determines the management category of the file. The possible category designations are:

- Mass storage file
- Scratch file
- Output file
- Drop file
- Write-temporary file

MASS STORAGE FILES

The originating user controls the creation and disposition of mass storage files. CYBER 200 OS protects mass storage files from access or destruction by other nonprivileged users.

SCRATCH FILES

Only a task can create scratch files. Scratch files exist during the originating task's activity. When the task terminates normally, all scratch files are destroyed. When the operating system terminates the task or the task terminates and saves its drop file, scratch files are saved. A CLOSE FILE system message specifying a scratch file destroys the file. All scratch files have read and write access.

Scratch files are not local files. Local files exist for the duration of the user job; scratch files exist only for the duration of the task.

OUTPUT FILES

Output files contain information suitable for processing by an output device, such as a printer, card punch, or microfilm device. Only a user task can create an output file. When one of the following occurs, CYBER 200 OS gives all output files with valid disposition codes to privileged system tasks for output processing:

- The task terminates normally.
- It issues a CLOSE FILE (f=0005) message.
- It issues a ROUTE AND FILE DISPOSITION (f=000D) message.
- It issues a TERMINATE (f=0006) message.

After output files are processed, they are destroyed. If the disposition code is not valid, the file remains as an unattached permanent file.

DROP FILES

CYBER 200 OS creates a drop file for each task called into execution. The executing task is called the source file. Its drop file contains modified pages of the source file, free space, and write-temporary files. Modified pages for other files are written directly to the respective file. Volume 1 gives further description of drop files and their naming conventions.

WRITE TEMPORARY FILES

A write-temporary file is a read-only file that has been modified while paged in to central memory. The modified pages cannot be paged back to the read-only file, and so are paged to the drop file. Subsequent references to those pages access the modified version from the drop file. To reference the original read-only version, the modified pages must be removed from the drop file.

FILE I/O

As described in volume 1, CYBER 200 OS performs two types of input/output, implicit and explicit.

The EXPLICIT I/O system message (f=0050) can perform explicit I/O. With a single system request, this message can transfer one or more blocks between the specified buffer and a storage device. However, the system must lock down the buffer in memory (it cannot be paged out while in use); therefore, more system action is required to prepare for explicit I/O than for implicit I/O.

With implicit I/O, information transfers directly between a storage device and its current location in central memory. The transfer occurs when the user causes an access interrupt by referencing a page of data or code not in memory. If the virtual page has been previously associated with physical space, the system transfers the data between memory and the physical device. If a virtual-to-physical relationship has not been established previously, the system defines the virtual page in free space so that it becomes an extension of program space.

When a task opens a file for implicit I/O, it can associate virtual and physical addresses using the MAP system message.

CYBER 200 OS recognizes two types of file addressing, virtual and physical.

PHYSICAL FILES

A physical file is accessed by physical addresses. It is, by definition, a data file. It cannot be executed. File I/O can be implicit or explicit. A physical file never has a minus page.

A virtual code file is a controllee file produced by the LOAD utility. Its first page is its minus page.

A virtual data file also has a minus page. It contains data accessed by virtual addressing. File I/O can be implicit or explicit.

MAGNETIC TAPE FILES

The CREATE FILE system message (f=0001) can access a tape file as a local file. The user specifies whether the tape is labeled or unlabeled and the volume serial number, recording density, and character code set (BCD or ASCII) for the tape.

The OPEN FILE system message (f=0003) can open the tape file for explicit I/O. Once opened, the task can give tape file access to its controllee with the GIVE TAPE ACCESS TO CONTROLLEE system message (f=#000C). When I/O to the tape file is complete, the CLOSE FILE system message (f=0005) can close the file.

The DESTROY FILE system message (f=0002) can sever a program's connection to a tape file and release the tape drive for reassignment. A tape file that has been closed, but has not yet been destroyed, can be reopened with another OPEN FILE message.

Programs issue system messages to direct CYBER 200 OS processing. With two exceptions, the system messages described in this section are calls to the virtual system. (The ADVISE and PROCESS SYSTEM PARAMETER messages are calls to the resident system.)

SYSTEM MESSAGE EXECUTION

A program can use either of two methods to issue a system message. It can call an SIL subroutine which, in turn, issues a system message, or it can issue the system message directly. SIL subroutines are described in volume 1.

To issue a system message directly, the user presets one or two blocks of words known as the Alpha and Beta of the message and then issues an exit force instruction. The Alpha and Beta formats for each message are referenced in the individual message descriptions. The exit force instruction is described in the CYBER 200 Hardware Reference Manual.

A 32-bit indirect or 64-bit direct pointer immediately follows the exit force instruction within the instruction stream. It points to the system message Alpha. When the exit force instruction is executed, system operation changes to monitor mode and the system message is executed.

The hexadecimal format of an indirect message pointer is:

00EE00rr

where rr is the number of the register containing the virtual address of the message. The hexadecimal format of a direct message pointer is:

00FFaddress

where address is the virtual address of the first full word of the message (12 hexadecimal digits).

When a message is processed without error, the operating system returns control to the half-word or full-word immediately following the message pointer. If that half-word or full-word contains another message pointer, the system executes that message without returning to job mode. Any number of system messages can be chained in this way.

ALPHA AND BETA WORD CONVENTIONS

System messages have a two-part standard format. The first part, called the Alpha portion, specifies the function to be performed, the length of the Beta portion, and where to proceed for error processing. The Alpha portion has the same general format for all messages, and is always either two or three words in length.

The second part, called the Beta portion, contains parameters and varies greatly in length from one message to the next. The format of the Beta portion depends on the function, as described later in this section for each function code. The user specifies in the Alpha portion what the length of the Beta portion is and, in some cases, where it is located. The message descriptions in this section specify what the minimum size of any particular Beta must be. The user can, however, specify a larger Beta in which case the extra space is simply left unchanged.

Alpha and Beta portions must start on full-word boundaries and cannot exist in the user's page zero; rather, they must exist in virtual space that has read/write or write-temporary access. Alpha and Beta portions must not cross large page boundaries.

NOTE

Options/control field values of #E0 through #FF, response code (r field) values of #7000 through #7FFF, and error response (ss or cerr field) values of #E0 through #FF are reserved for installation use. The options/control field is the eight bits to the left of the function code field in Alpha. The response code field is bits 1 through 15 in Alpha (1), where bits are numbered from zero.

Values returned in the r, ss, and cerr fields are hexadecimal.

In the figures in this section, some of the Alpha and Beta words are drawn with dashed lines. This is meant to indicate that these words are optionally present.

OVERVIEW

The following are the available system messages listed according to functional areas. (The comments in parentheses clarify the purpose of the message.)

File Management

Create File (f=0001)
Attach File (f=0010)
Change File Attributes (f=#000B)
Give File (changes file ownership) (f=0008)
Destroy File (f=0002)
Pool File Manager (f=0026)
Route and File Disposition (f=#000D)

Input/Output Operations

Open File (f=0003)
Give Tape Access to Controllee (f=#000C)
Map (into virtual space) (f=0004)
Explicit I/O (f=0050)
Give Up CPU Until I/O Completes (f=0052)
Close File (f=0005)

Interrupt Processing

Abnormal Termination Interrupt (f=0020)
Program Interrupt (f=#001C)
Return From Interrupt (f=0051)

Starting and Ending Program Execution

Execute IQM Request (f=0030)
Execute Program for User Number (f=0022)
Recall (suspends program execution) (f=0025)
Terminate (ends program execution) (f=0006)

Controllee Chain Processing

Initialize Controllee Chain (f=#001D)
Initialize or Disconnect Controllee (f=#001B)
List Controllee Chain (f=0013)
Remove Controllee from Main Memory (f=0019)

Message Communication

Get Message from Controllee (f=0017)
Get Message from Controller or Operator (f=0016)
Send Message to Controllee (f=0015)
Send Message to Controller (f=0014)
Send Message to Operator (f=#001A)
Send Message to Dayfile (f=0029)
Message Control (redirects messages) (f=0018)

File Space Allocation

Advise (on virtual space requirements) (f=#FF00)
Process System Parameter (sets memory limits) (f=#FF01)
Reduce File Length (releases unused space) (f=#000A)

Information Retrieval

Get Pack Label and PFI (f=0011)
List File Index or System Table (f=0009)
List Unformatted File Index Table (f=0007)
Miscellaneous (f=0024)

Accounting

Update User Directory (f=0023)
User/Accounting Communication (f=#000E)
Variable Rate Accounting (f=0028)

Special Functions

Execute Operator Command (for operator user number) (f=0021)
Link (link station functions) (f=0027)

MESSAGES

The message descriptions in this section are in function code order. Table 4-1 lists the messages in alphabetical order.

TABLE 4-1. MESSAGE FUNCTION CODES

Message	Hexadecimal Function Code†	Message	Hexadecimal Function Code†
ABNORMAL TERMINATION INTERRUPT	0020	LIST FILE INDEX OR SYSTEM TABLE	0009
ADVISE	FF00	GET PACKLABEL AND PFI††	0011
ATTACH	0010	GIVE FILE	0008
CHANGE FILE ATTRIBUTES	000B	GIVE TAPE ACCESS TO CONTROLLEE	000C
CLOSE FILE	0005	GIVE UP CPU UNTIL I/O COMPLETES	0052
CREATE FILE	0001	INITIALIZE CONTROLLEE CHAIN	001D
DESTROY FILE	0002	INITIALIZE OR DISCONNECT CONTROLLEE	001B
EXECUTE IQM REQUEST††	0030	LINK††	0027
EXECUTE OPERATOR COMMAND	0021	LIST CONTROLLEE CHAIN	0013
EXECUTE PROGRAM FOR USER NUMBER††	0022	LIST UNFORMATTED FILE INDEX TABLE	0007
EXPLICIT I/O	0050	MAP	0004
GET MESSAGE FROM CONTROLLEE	0017	MESSAGE CONTROL	0018
GET MESSAGE FROM CONTROLLER OR OPERATOR	0016		

TABLE 4-1. MESSAGE FUNCTION CODES (Contd)

Message	Hexadecimal Function Code†	Message	Hexadecimal Function Code†
MISCELLANEOUS	0024	ROUTE AND FILE DISPOSITION	000D
OPEN FILE	0003	SEND MESSAGE TO CONTROLLEE	0015
POOL FILE MANAGER	0026	SEND MESSAGE TO CONTROLLER	0014
PROCESS SYSTEM PARAMETER	FF01	SEND MESSAGE TO DAYFILE	0029
PROGRAM INTERRUPT	001C	SEND MESSAGE TO OPERATOR	001A
RECALL	0025	TERMINATE	0006
REDUCE FILE LENGTH	000A	UPDATE USER DIRECTORY††	0023
REMOVE CONTROLLEE FROM MAIN MEMORY	0019	USER/ACCOUNTING COMMUNICATION	000E
RETURN FROM INTERRUPT	0051	VARIABLE RATE ACCOUNTING††	0028

1E, 1F, E0 through FF (reserved for installation use).
†Rightmost field in Alpha(1). Abbreviated as f.
††Available to a privileged task or USER1 task only.

CREATE FILE (f=0001)

The CREATE FILE message gives a user space on a mass storage device, names that space, and defines parameters for it. The operating system makes an entry in the file index table and pack file index for this named space (that is, file), and initializes fields in the entry using information given in the message.

A CREATE FILE message effectively creates and opens a file. An OPEN FILE message is necessary only for accessing an existing file that has been closed.

After CREATE FILE message processing, sufficient information is stored in the user program's minus page for the user program to perform either implicit or explicit input/output on the file just created. The user must set the mode field in the CREATE FILE message to select the form of input/output (implicit or explicit). Either form of input/output can be performed on either type of file (physical or virtual).

Implicit input and output can be performed on a file just created only after a correspondence of virtual space to physical mass storage has been established. This can be done by means of either of the following:

- The user specifies a virtual base address in the bva field of the CREATE FILE message.
- The user issues a MAP message to establish the correspondence.

Explicit input and output can be performed immediately after the file has been created with the mode field specifying explicit input/output.

A privileged user can set some of the values in the new file index table entry the operating system creates every time a file is created. Five Beta words are required for a privileged create. In Beta(5) the user can indicate a formatted or unformatted version of the file index entry. For the formatted option, Beta(5) contains the virtual bit address of a file index table entry copy formatted as shown for the option 1 Beta portion of the LIST FILE INDEX OR SYSTEM TABLE message. The following fields of the copy must be set by the user, as the operating system takes whatever values are in these fields and uses them to initialize the file's file index table entry: ref, rep, lodlen, type, slev, mcat, acs, lok, shacc, fact, tid, dc, ic, ec, cm, ot, zip, st, and fipri. The system also returns values to the following fields in the copy: saddr, unit, wlen, act, ext, origlen, seglen, segadr, dorg, dola, dolm, torg, tolm, and tlr. All other fields shown for the LIST FILE message are not used.

For the unformatted option, Beta(5) contains the virtual bit address of a file index table entry copy as shown for the file index table in section 2 of this manual. The user sets the following fields of the copy, which the system uses to initialize the created file's file index table entry: ref, rep, lodlen, type, slev, mcat, acs, shacc, fact, tid, dc, ic, ec, cm, ot, zip, st, fipri, origlen, sfo, bt, mnr, mxr, rt, pc, and rmk. The system returns values to all the other fields in the copy.

The CREATE FILE message is also used to gain access to a magnetic tape (mcat=4). Four Beta words are required (privileged creates are not allowed for tape files). Beta(4) is formatted differently than for mass storage files and contains parameters unique to tape. If a requested tape is not currently known to the operating system, the system issues a message requesting the operator to mount the tape. The program issuing the tape request is set at the point of reissuing the CREATE FILE message and is then removed from main memory. The message is retained until the operator responds (either with an ASSIGN or with an ABORT TAPE REQUEST). Only one tape request message can be retained at any one time for each executable program. If more than one tape has been requested as a result of a CREATE FILE message, and if more than one of the tapes is unknown to the system, only the last of the tape requests is retained by the system for operator response. When the operator makes the tape known to the system, an entry is made in the file index table to describe the tape file. The physical unit number is returned in the unit field of the Beta for magnetic tape files. The system then reactivates the program.

During creation of a file to be opened for implicit input and output (mode=1) and for which extensions are allowed (ext=0), a virtual address overlap error (ss=16) can occur if not enough virtual space exists for the extension between the end of this file and the beginning of the next.

When the station message processor (SMP) issues a CREATE message to create a link file (mcat=8), a zip code and CYBER 200 mainframe identifier are returned in the length and bva fields of Beta(4). When a privileged utility or some other privileged user issues a CREATE message with mcat=8, the user must set up a parameter package that defines the type of link file, and place the length and virtual bit address of the package in the length and bva fields of Beta(4). A nonprivileged user cannot issue a CREATE with mcat=8. The parameter package for defining the type of link file is shown with the format of the CREATE FILE message in figure 4-1. (The Beta portion of the message can actually consist of more than one of the four- or five-word sets shown in the figure.)

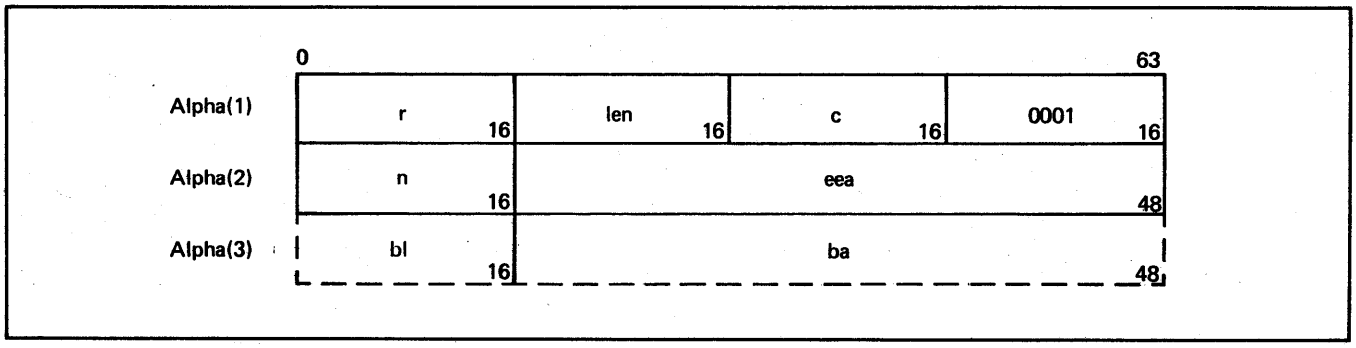
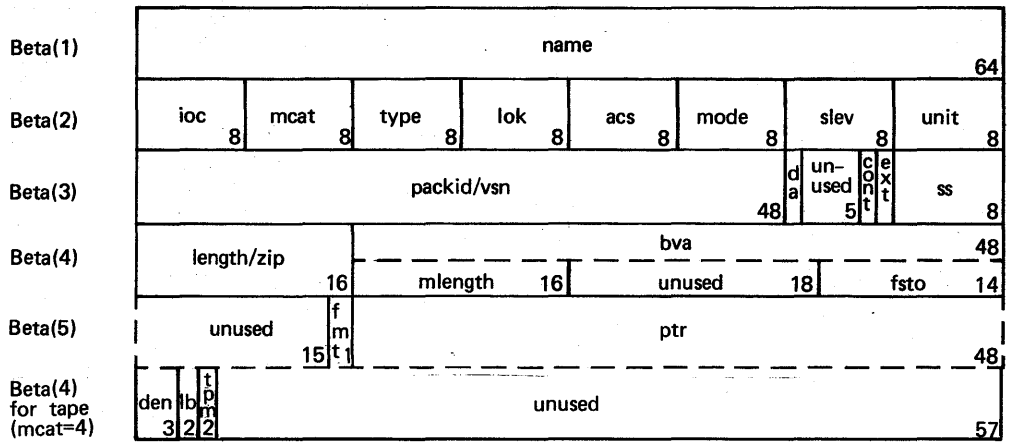


Figure 4-1. CREATE FILE Message Format (Sheet 1 of 6)



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in an ss field of Beta(3)
2	User is not privileged (privileged creates only)
211	Number of creates in this message is illegal (the n field is zero or greater than 16)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. All requests must provide at least four Beta words; and for privileged creates, len must be a multiple of 5.

c — Create mode:

0	Request a local file
1	Define an unattached permanent file (privileged only)
2	Define a permanent file (make local file permanent or create a permanent file)

n — Number of creates in this message; maximum is 16.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

name — File name, in ASCII, left-justified with blank fill. File names must be in the format described in section 3. This parameter is not required when the mcat field is 5.

ioc — The file's input/output connector number (#0 to #F, #12 to #47); or #FF, which causes the operating system to allocate an input/output connector and then to return the number in this field. If no input/output connector is available, an error code of 1D is returned in the error response field. This field is set by the operating system when the mcat field is 5.

mcat — File management category:

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
4	Magnetic tape file
5	User-generated drop file; return the drop file name in Beta(1)
7	Batch file
8	Link File (privileged use only)
A	Checkpointed output file
F	Checkpointed input file

The operating system sets the mcat field to zero for a privileged create. For categories 0 through 4 of this field, standard file name conventions apply. For category 5, the drop file name returned begins with a number (which is standard for drop file names); if a new drop file is created, the existing system-created drop file is destroyed.

Figure 4-1. CREATE FILE Message Format (Sheet 2 of 6)

type — Type of device or file:

0	Physical file
1	Virtual data file
2	Virtual code file
7	7-track tape
9	9-track tape

The operating system sets this field to zero for privileged creates, and sets it to 1 when the mcat field is 5.

lok — File access lockout. The values are:

0	No access is locked out (same as lok=4)
1	Write access is locked out (same as lok=5)
2	Read access is locked out (same as lok=6)
3	Read and write access are both locked out (same as lok=7)

The operating system locks out read and write access on privileged creates.

acs — Default file access permission. The values are:

0	No access is allowed (same as acs=4)
1	Write access is allowed (same as acs=5)
2	Read access is allowed (same as acs=6)
3	Read and write access are both allowed (same as acs=7); not allowed if labeled tape

The operating system sets access permission to read access on privileged creates, and sets it to 3 when mcat field is 5.

mode — Input/output mode. If this field is zero, the operating system opens the file for explicit input and output. If the field is 1, the operating system opens the file for implicit input and output. The operating system sets this field to 1 if the mcat field is 5.

slev — Security level, 1 to 255, to be given to the file if this field is nonzero and greater than that of the task issuing this message. If the field is zero or less than or equal to the security level of the caller, the security level belonging to the caller of this message is used.

unit — A logical unit number is returned in this field for mass storage files; a physical unit number is returned for tape files.

packid — Six-character identifier of the disk pack on which the file is to be created. When the call is made and this field is binary zero, the operating system creates the file on a system pack where space is available, and returns the packid of the pack to this field. This parameter is ignored when the mcat field is 4 or 5.

vsn — Volume serial number; a six-character identifier for tape, displayed as part of the load request. When the call is made and this field is binary zero, the volume serial number of the tape assigned is returned in this field. This parameter is used for tape files only (mcat=4).

da — Define action returned by the system:

da=0	File created (c=2 only)
da=1	Local file made permanent (c=2 only)†

cont — File contiguity. This field must be set to zero if the file to be created does not necessarily have to be contiguous (is in two segments), or set to 1 if the file must be contiguous. If this field is zero and the file is created contiguously, the operating system sets the cont field in the Beta word to 1, but the file index table remains as set by the user. The operating system sets this field to zero if the mcat field is 5.

ext — File extendability. This field must be set to zero if the file is to be created with extensions allowed, or set to 1 if the extensions are not allowed. The operating system sets this field to zero if the mcat field is 5.

ss — Error response field. The values are:

1	File already exists
2	No available mass storage space for this file
3	Illegal value in the mcat field
4	Parameter or format error occurred
5	Operator-initiated tape error occurred
6	Input/output connector is already in use
7	File index table is full
8	Standby job cannot create a tape
9	Invalid file name
A	The ss field was preset
B	Existing drop file cannot be destroyed; the system table DMAP is full or the file is open to another problem program

Figure 4-1. CREATE FILE Message Format (Sheet 3 of 6)

C	Illegal value in the type field
D	For mcat=5, pages have been written already to the existing drop file; no new file was created
E	For mcat=5, drop file length is not adequate to hold space already in the drop file map
F	Unable to find the requested pack identifier
11	Cannot send a tape request message to the operator
14	Requested file size is greater than installation parameter LDSK
15	Bound implicit map area in the minus page is full
16	Virtual address overlap
17	Could not find a slot in the pack file index for the file index table entry
18	Could not make a pack file index entry
19	Tried to create a virtual file but the file is less than two sectors long
1A	Disk is logically turned off
1B	Cannot locate the user or pool (privileged creates only)
1C	No room for file entries (privileged creates only)
1D	No input/output connector is available
1E	No FST space available
1F	Attempt to create a file for implicit I/O with write only access
21	For mcat=8, mainframe (ST) not logged in
22	For mcat=8, privileged use only
23	User permanent file space exceeds installation limit
24	Number of user files exceeds installation limit
25	No drop file pack found (drop file creates only).
26	Attempt to DEFINE a tape file

length/zip — The value in this field depends on what the value in the mcat field is. If the length/zip field contains the length of the file in small pages (maximum length is an installation parameter, LDSK). If the user is privileged and the mcat field is 8, the length/zip field contains the length of the parameter package whose address is in the bva field. If the user is SMP and mcat=8, the length/zip field contains a zip code returned by the operating system. If mcat=5 and length is less than or equal to the efficient run size (ERS), the length/zip field contains the length of the parameter package whose address is in the db field.

bva — The value in this field depends on who issued the message and what the value in the mcat field is. If the user is non-privileged, the bva field contains the virtual bit address of the first word of the file being created. The file is not mapped in if the field is zero; a subsequent MAP (function 4) call is required.

If the user is the SMP, the field contains the CYBER 200 mainframe identifier; the format is three ASCII characters, right-justified.

If the user is privileged and the mcat field is 8, the bva field contains the virtual bit address of the first word of the parameter package that describes the link file being created. The length of the parameter package must be given in the length/zip field. The parameter package formats are given in table 4-2.

mlength — Maximum length in small pages to which the file can be extended. If the file is not created with write access, mlength=length is returned by the system. If the file is created with write access, the system returns mlength=length + (length *EXTSIZ/100), where EXTSIZ is an installation parameter.

fsto — File segment table (FST) ordinal; for privileged users only. Returned by the operating system if the create is successful.

fmt — For privileged creates, this bit describes the format of the area pointed to by ptr.

0	Formatted file index entries as shown for option 1 of the LIST FILE INDEX OR SYSTEM TABLE message, f=0009.
1	Unformatted file index entries (same as described for the file index table in section 2)

den — Magnetic tape density.

0	200 bpi (7-track)
1	556 bpi (7-track)
2	800 bpi (7- or 9-track)
3	1600 bpi (9-track)

lb — Tape label format.

0	Standard ANSI labels
1	Unlabeled

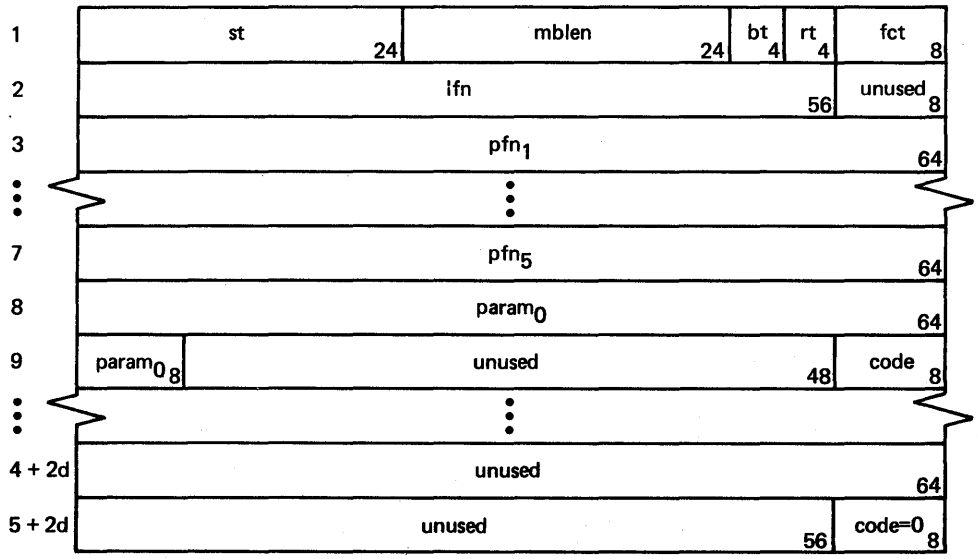
tpm — Tape mode.

0	External BCD (7-track)
1	Unformatted binary/8-bit ASCII (7- or 9-track)
2	6-bit ASCII (7-track)

Figure 4-1. CREATE FILE Message Format (Sheet 4 of 6)

ptr — For privileged creates, this field contains the virtual bit address (furnished by the user) of the first word of a 16-word copy of the file index table entry. As explained in the main text, the system uses some of the values in the copy to initialize the file's file index table entry, and returns values to certain other fields in the copy. The format of this file index table entry copy is the same as for the Beta portion of the LIST FILE INDEX OR SYSTEM TABLE message (f=0009), option 1, or unformatted as described in section 2, depending upon the setting of fmt.

Parameter package for staging a file:



st — A three-character mainframe identifier (set by the caller).

mblen — The maximum block length, in bits.

bt — CYBER block type:

2	C type (character count)

rt — CYBER record type:

3	Z type (zero-byte delimited record)
8	S type (semi-logical record)

fct — Function to be performed. Hexadecimal values are:

8	GETPF
10	SAVEPF
20	PURGE

lfn — Logical file name (maximum of 7 characters, left-justified with blank fill).

pfn — Permanent file name; 1 to 8 letters or digits, the first of which must be a letter, left-justified with blank fill.

param — Parameters for the function specified in the fct field. ASCII data, right-justified with binary zero fill. One or more parameters can be listed each with a second word containing the right 8 bits of param and a code that indicates what kind of data is in the param field (see code field description).

code — The type of the value in the param field. The possible values for this field are listed in table 4-2. A code of 0 must be present in the second word following the last two-word parameter entry.

Figure 4-1. CREATE FILE Message Format (Sheet 5 of 6)

Parameter package for spooling a file:

1	st 24	unused 24	type 8	fct 8	
2	lfn 56			unused 8	
3	sid 24	did 24	tid 16		
4	status 8	0 6	length 18	fc 16	dc 16
5	unused 32		ic 16	ec 16	
6	account 64				

st — A three-character mainframe identifier (set by caller).

type — Disposition:

- 1 Print
- 2 Punch

fct — Function to be performed:

- 3 Spool in
- 4 Spool out

lfn — Logical file name (binary).

sid — Originating mainframe identifier; same as the st field for input.

did — Destination mainframe identifier; same as the st field for output.

tid — Terminal identifier; if zero, indicates the central site.

status — Dayfile:

- 0 No dayfile with file
- 2 Dayfile exists in last record of the file

account — Account number (ASCII, left-justified with blank fill).

length — File length in small pages.

fc — Reserved for forms code.

dc — Disposition code. See LIST FILE INDEX or SYSTEM TABLE message (figure 4-10).

ic — Internal characteristic. See LIST FILE INDEX or SYSTEM TABLE message (figure 4-10).

ec — External characteristic. See LIST FILE INDEX or SYSTEM TABLE message (figure 4-10).

---

†When a local file is made permanent, the following beta field is valid: name. The following beta fields are ignored: fsto, ptr, type, lok, acs, mode, slev, unit, packid/vsn, cont, ext, len, bva, mlen, and ioc.

Figure 4-1. CREATE FILE Message Format (Sheet 6 of 6)

TABLE 4-2. PARAMETERS FOR LINK FILE CREATE (MASS STORAGE FILE ONLY)

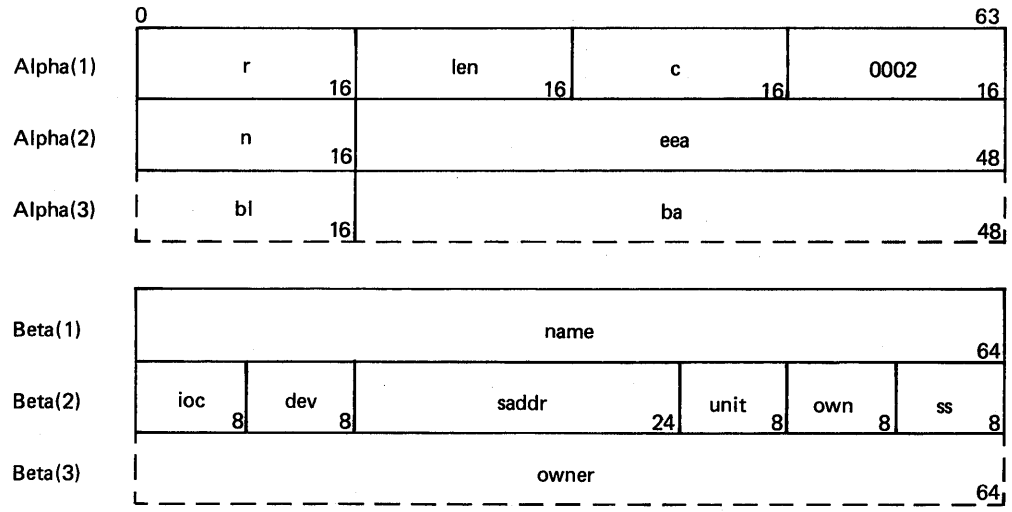
Code Field	Keyword Parameter	Significance
0	none	End of FDB list.
2	RP	Retention period (days).
3	CY	Cycle number.
4	TK	Turnkey password, in ASCII.
5	CN	Control password, in ASCII.
7	MD	Modify password, in ASCII.
8	RD	Read password, in ASCII.
9	MR	Multiread indicator.
B	XR	Control, modify, extend password, in ASCII.
C	ID	Owner's user number, in ASCII.
E	AC	Account number, in ASCII.
F		Reserved.
10	PW	Control password, in ASCII.
11	PW	Modify password, in ASCII.
12	PW	Turnkey password, in ASCII.
13	PW	Extend password, in ASCII.
14	PW	Read password, in ASCII.
15		Reserved.
19	LC	Lowest cycle number.
1A	ST	Staging identifier, in ASCII.
20		Reserved.
21		Reserved.

The operating system always sets the ioc, type, acs, mode, and slev fields in the Beta portion of the message when the file being created is a user-generated drop file (mcat=5). The operating system also sets the mcat, type, and acs fields for a privileged create (c=1); otherwise, values of the message fields are provided by the user.

DESTROY FILE (f=0002)

The DESTROY FILE message can be issued to sever a program's connection with a tape drive and release the drive for reassignment, or to sever the program's connection with a mass storage file and release the mass storage space. At the conclusion of DESTROY FILE message processing, any mass storage file referenced by the message has ceased to exist, as have any modified pages of the file. Virtual address definitions pertaining to this file are no longer defined, and the input/output connection and map entries are erased. For nonprivileged destroys only, a file need not be closed before the DESTROY FILE message is issued. The format of the message is shown in figure 4-2. (The Beta portion of the message can actually consist of more than one of the 2- or 3-word sets shown in the figure.)

If a mass storage file is at a sufficiently high security level, it is overwritten with a pattern when it is destroyed. Some installations can choose to overwrite all files when they are destroyed. A privileged destroy is not a close and destroy, as is the nonprivileged destroy; the privileged destroy must be preceded by a privileged close.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in an ss field of Beta(2)
2	User is not privileged (privileged destroys only)
211	Number of destroys in this message is illegal (the n field is zero or greater than 16)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. The value of len must be a multiple of 2 (for regular destroys) or of 3 (for privileged destroys).

c — Destroy mode:

0	Return local and attached permanent files
1	Privileged purge of a permanent file
2	Purge of a permanent file (makes file local if attached)
3	Purge of a poolfile in pool in Beta(3)

n — Number of requests in this message; maximum is 16.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

name — File name, in ASCII, of the file to be destroyed. File names must be in the format described in section 3.

Figure 4-2. DESTROY FILE Message Format (Sheet 1 of 2)

ioc — If dev=4, this field must contain the file's input/output connector number. If dev=0 and the file is a tape file, the ioc number need not be supplied by the user and is returned by the system if the file was open. If a mass storage file is being destroyed, the operating system returns, in this field, the inclusive-OR of all input/output connector numbers connected to this file.

dev — Device type:

0	Mass storage device or magnetic tape device
4	Magnetic tape device (optional, see ioc)
8	Reserved for CYBER 200 Link Station file

saddr† — Beginning sector address of the file to be destroyed. This field is significant only when the unit field is nonzero. This field should be set to zero when a nonprivileged user issues this message.

unit† — Logical unit number of the file to be destroyed. This field is significant only when the saddr field contains a valid sector address. This field should be set to zero when a nonprivileged user issues this message.

own — Ownership of the file to be destroyed. This field is significant only for nonprivileged users (C≠1). The values

0	Private ownership
1	Public ownership; valid only for privileged users
2	Pool ownership; valid only for the pool boss

ss — Error response field. The values are:

0	Normal completion
1	File name does not exist
2	File name given is in conflict with that in the input/output connector
3	Another active program has the file open, or the file has been privileged-opened
4	Unused
5	Nonprivileged task tried to destroy a public file
6	User other than the pool boss tried to destroy a pool file
7	Illegal input/output connector number specified
8	Drop file map is full
9	Error trying to remove the pack file index entry
A	Disk is logically off
C	No room in FILEI for privileged destroy pseudo-logon
F	Cannot destroy a public file unless privileged
10	Attempt to purge a tape file

owner — For privileged destroys, a user number or pool name to which the file being destroyed belongs. The binary user number must be right-justified with zero fill, or, if this is the pool name, it must be left-justified with blank fill.

†Fields saddr and unit are inspected by the system only when a nonprivileged destroy is made by a USER1 routine.

Figure 4-2. DESTROY FILE Message Format (Sheet 2 of 2)

OPEN FILE (f=0003)

Mass Storage Files

The OPEN FILE message connects the user's program to a preexisting mass storage file for performing input and output on the file. In opening a file, the user can accept the parameters given to the file when it was created; or else, if the file owner has given permission, the user can alter the parameters. Both physical and virtual files can be opened for either explicit or implicit input and output. Once opened for explicit input and output, however, a file cannot be accessed implicitly, and vice versa. Nevertheless, a file can be open in several input/output connectors at the same time - some for implicit input and output, and others for explicit input and output.

When a program opens a physical file in explicit mode, the specified input/output connector in the program's minus page is filled in as required and an entry is made in the explicit file map area of the minus page. This allows initiation of explicit input and output. In this mode, the file is accessed by explicit requests to transfer data into buffer areas. The EXPLICIT I/O message (f=0050) must be used to define the buffers and initiate data transfers.

When a program opens a physical file in implicit mode, the specified input/output connector in the program's minus page is completed and an entry is made in the implicit map area of the program's minus page. No entry is made in the bound explicit map. Explicit input/output cannot be accomplished on a physical file that is opened in implicit mode. A physical file opened in implicit mode is considered to begin at the virtual address given in the working virtual address field of Beta(4). The virtual address space represented by the file is considered to be contiguous over the entire length of the file beginning at file word zero. When the OPEN FILE message has been processed, the user can map out the space just defined and map in the file in any manner. All implicit input/output attributes normally pertaining to virtual files are applied to physical files in implicit mode.

When a program opens a virtual file in explicit mode, all input/output must be done explicitly through the program's buffers in the same manner as for physical files opened in explicit mode. The input/output connector number specified in the program's minus page is filled in, and one entry is made in the explicit map. The file is mapped, beginning with word zero of the virtual file's minus page. When a file is opened in explicit mode, no implicit access is possible to any of the virtual space usually represented by the file.

When a program opens a virtual file in implicit mode, the input/output connector number in the program's minus page is filled in; and, optionally, implicit map entries are completed. If the user chooses to use the file maps as recorded in the minus page of the disk copy of the file, the entries are simply copied to the bound implicit maps. Alternatively, the user can choose to have the file maps copied into the program buffer; in this case, only the input/output connector is filled in, no implicit map entries are made, and the program cannot yet access the file implicitly. This kind of open call should be succeeded by a map-in call that tells the system how to relate virtual space to the physical disk file. A user also can map a file into contiguous space (beginning at word zero of the minus page) according to parameters supplied with the call.

For privileged opens to occur, the file must not be open with write access by anyone; while the file is privileged open, all attempts to open with write access are barred. If the c1 field in Beta(2) is 1, these rules are extended to exclude an open of any sort to assure that the privileged open is successful.

A privileged user can fetch a copy of the opened file's file index table entry by specifying a virtual bit address in Beta(5). The copy is returned beginning at the specified address. This copy is not used in the same way that the copy can be used on a privileged create; initializing fields in the copy associated with an OPEN FILE message does not alter the values in the file index table entry.

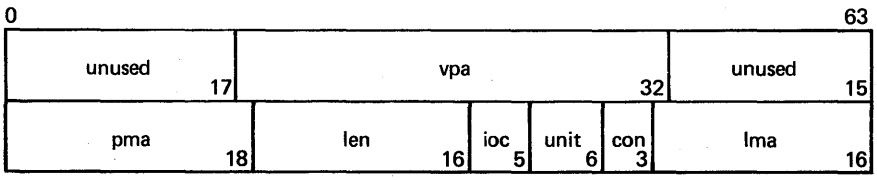
During opening of a file in implicit mode for which extensions are allowed (ext=0), a virtual address overlap error (ss=23) could occur if not enough virtual space exists for the extension between the end of this file and the beginning of the next one. If the file was created with no extensions allowed, it cannot be opened with extensions allowed; however, if the file was created with extensions allowed, it can be opened with either extensions not allowed or extensions allowed.

A program requesting delivery of an implicit map (by means of a map field value of zero or 1) must supply a buffer to hold the entire map. Although the operating system does not store entries beyond the end of the buffer supplied, neither does it squeeze out empty map entries. The format of the map returned to the program buffer is two words per map entry, as shown in figure 4-3.

Magnetic Tape Files

The OPEN FILE message is also used to connect the user's program to a previously requested tape. A tape file can only be opened for explicit input and output.

When a tape file is opened, an input/output connector in the program's minus page is completed. The appropriate correspondence table entry is updated with the assigned ioc number and the descriptor block number of the requesting program. A privileged open of a tape file is not vaild.



vpa Virtual page address.

pma Physical mass storage sector address.

len Length of this segment in small page blocks.

ioc Input/output connector number.

unit Unit.

con Control.

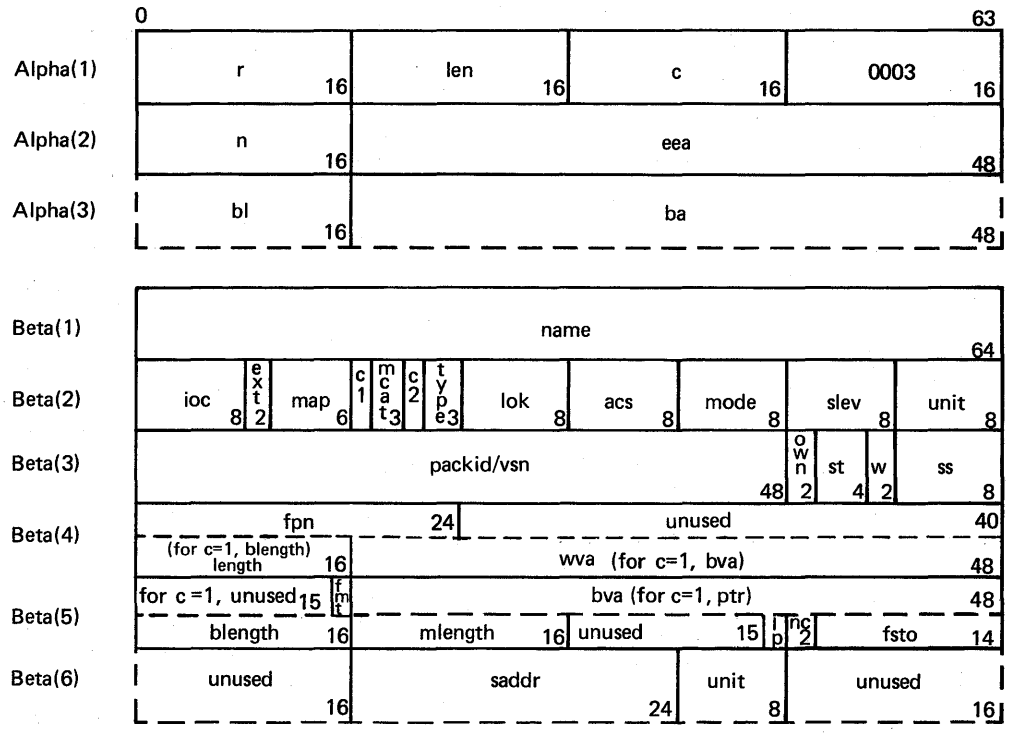
lma Logical mass storage sector address.

(For further definition of the unit, con, and lma fields, see Bound Implicit Maps in section 2.)

Figure 4-3. Format of Bound Implicit Map Entry in Program Buffer

Note that only the following Beta fields are valid for a magnetic tape file: name, ioc, meat, c2, type, lok, acs, mode, slev, unit, vsn, own, st, ss, lp, and fpn.

The format of the OPEN FILE message is shown in figure 4-4. (The Beta portion of the message can actually consist of more than one of the 5- or 6-word sets shown in the figure.)



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in an ss field of Beta(3)
2	User is not privileged (privileged opens only)
211	Number of opens in this message is illegal (the n field is zero or greater than 16)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. This field must be a multiple of 6 if the c field is 2; this field must be a multiple of 5 if the c field is zero or 1.

c — Open mode:

0	Regular open (file must be attached)
1	Privileged open (fields dola, tlr, dolm, and tolm in the file index table are not updated for the file being opened)
2	Regular open; can be issued by USER1 routines only (system also attaches the permanent file)

This field determines the settings of several other fields in Beta(2) of this message. When this field is zero, the c1 and c2 options enable the user to modify fields in the file index table. Permission to modify these fields is granted by the system if the file ownership is: private; pool, and the user is the pool boss; or public, and the user is privileged. When this field is 1, the c1 option enables the privileged user to specify who can access the file for the duration of this open, and the c2 option is not used.

n — Number of files to be opened at this time; maximum is 16. At times, it might be more efficient to open more than one file at a time. When this is to be done, the Alpha portion for the OPEN FILE message is used once, with n equalling the number of files to be opened; this is followed by groups of Beta words, one group per file.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

Figure 4-4. OPEN FILE Message Format (Sheet 1 of 4)

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full words of the Beta portion.

name — File name, in ASCII. File names must be in the format described in section 3. If the format is not proper, error response 21 is returned in the ss field.

ioc — The file's input/output connector number (0 to #F, #12 to #47); or #FF, which causes the operating system to allocate an input/output connector and then to return the number in this field. If no input/output connector is available when the system attempts to allocate one, the system returns an error code of 37 in the error response field in this message.

ext — File extendability; set by the user. The values are:

- 0 — Extensions allowed if extensions were not prohibited on creation of the file (same as ext=2)
- 1 — No extensions allowed (same as ext=3)

Value which was set at creation time is returned by the operating system after a successful open. This field is zero if extensions were allowed or set to 1 if not allowed. If the file was created with no extensions allowed, it could not have been opened with extensions allowed; however, if the file was created with extensions allowed, it can be opened with either extensions allowed or not allowed.

map — Determines the disposition of maps residing in the minus page of the virtual file being opened for implicit input and output. For privileged opens, the operating system sets this field to 2. The values are:

- 0 — Obtain from the file's minus page the mapping information that is to be inserted into the bound implicit map area of the program's minus page.
- 1 — Read mapping information from the file's minus page and return it to the buffer specified in Beta(5). An input/output connector is filled in, but no entries are made into the bound implicit map area of the program's minus page. The file is not mapped in.
- 2 — Do not read mapping information from the file's minus page; obtain mapping information according to the w option of this message.

A program requesting delivery of the implicit maps must supply a buffer (figure 4-3) to hold the entire map; buffer length and address are in Beta(5).

c1 — Open the file as specified in the mode field. For regular opens, the values are:

- 0 — Do not change the file type
- 1 — Change the file type to the one in the type field

For privileged opens, the values are:

- 0 — Other privileged and nonprivileged opens are allowed, but without write access
- 1 — No other opens are allowed until the privileged open is complete; the privileged open cannot occur if any other opens currently exist

mcat — File management category to be associated with the file. For privileged opens, the operating system sets this field to zero. For regular opens, this field is copied into the mcat field of the input/output connector and the values are:

- 0 — Mass storage file
- 1 — Scratch file
- 2 — Output file
- 3 — Write-temporary file
- 4 — Magnetic tape file

c2 — Open the file as specified in the mode field. This option is not used for privileged opens. For regular opens, the values are:

- 0 — Do not change the file access
- 1 — Change the file access field in the file index table to that specified by the acs and lok fields

type — File type. If the c1 option is zero, the operating system returns the file type to this field. If the c1 option is 1, the file type is to be changed to the type specified by this field, which can be one of the following:

- 0 — Physical data
- 1 — Virtual data
- 2 — Virtual code
- 6 — 9-track tape (returned only, cannot be changed with c1 option)
- 7 — 7-track tape (returned only, cannot be changed with c1 option)

The operating system sets this field to zero for privileged opens.

Figure 4-4. OPEN FILE Message Format (Sheet 2 of 4)

lok — File access lockout value to be set in the file index table if the c2 option is 1. If the c2 option is zero or if a public file is opened, the operating system sets this field with the value from the lok field in the file index table. Whatever value is set in this field is also set in the lok field of the input/output connector for the file. The values are:

0	No access is locked out (same as lok=4)
1	Write access is locked out (same as lok=5)
2	Read access is locked out (same as lok=6)
3	Read and write access are both locked out (same as lok=7)

The operating system sets this field to 5 for privileged opens.

acs — Default file access permission. A logical conjunction is performed using the complement of the lok field in the file index table and this field as operands; the result returned to this field and to the acs field of the input/output connector for this file determines the access granted for this OPEN FILE call. If the c2 option is 1, the result is also stored in the acs field of the file index table. The values are:

0	No access is allowed (same as acs=4)
1	Write access is allowed (same as acs=5)
2	Read access is allowed (same as acs=6)
3	Read and write access are both allowed (same as acs=7). Not allowed for labeled tape files.

The operating system sets this field to 2 for privileged opens.

mode — Input/output mode. This field is zero if the file is to be opened for explicit input and output, or set to 1 if the file is to be opened for implicit input and output.

slev — Security level of this file, 0 to 255; set by the operating system.

unit — Logical unit number of the device on which the opened file resides; set by the operating system. The physical unit number is returned for tape files.

packid — Six-character identifier of the pack on which the file resides; set by the operating system.

vsn — Volume serial number of tape file; set by the operating system.

own — File ownership; set by the operating system. The values are:

0	Private
1	Public
2	Pool

st — Management category of the file; set by the operating system. The values are:

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
4	Magnetic tape file
5	Drop file created by the user
6	Drop file created by the operating system
7	Batch file

w — Location of the working virtual bit address to be used by the operating system when a physical file is opened for implicit input and output. If the map field is 2, this field is also used when opening a virtual type file for implicit input and output. If the virtual bit address to be used is zero, the file is not mapped in; a subsequent call to MAP (f=0004) is required. The values are:

0	Working virtual bit address is specified in the wva field of this message, and the file size is specified in the length field of this message. If the size is zero or greater than the length value given in the file index table, use the length specified in the file index table.
≠0	Working virtual bit address and length to be used are in the file index table. They are returned to the wva and length fields of this message by the operating system.

The operating system sets this field to zero for privileged opens.

ss — Error response field. The values are:

0	Normal completion
21	Either no name was given, or the file is not attached
22	Illegal value in the mcat field
23	Virtual address overlap
24	Input/output connector is already in use or not #0 through #45
25	Illegal value in the lok, acs, or type field
26	Mass storage error occurred while reading the minus page

Figure 4-4. OPEN FILE Message Format (Sheet 3 of 4)

27	Bound implicit map area in the minus page is full
28	File requested has a security classification higher than that of the caller
29	The sum of the bva and blength fields is greater than the maximum user virtual address
2A	Disk is logically off
2B	User directory was not found or the pool was not found (privileged opens only)
2C	Read or write open is not allowed; the file has been privileged-opened by another user
2D	Nonprivileged user
2E	Format error in a privileged open
2F	No more write opens permitted
30	The ptr field points to a file index copy that is out of bounds (privileged opens only)
31	No more room for the user table (privileged opens only)
32	Cannot open an attached file (privileged opens only)
33	No file segment table (FST) space available
36	More than one virtual address overlap; usually indicates an uninitialized minus page
37	No input/output connector available
38	Need six Beta words for option c=2
39	Illegal user number for option c=2
3A	Attempt to implicitly open a file with write only access
3B	Cannot privilege open tape file
3C	Cannot locate tape volume
50	Error in modifying the pack file index entry for this file

fpn — The file position number of the last file processed on a multifile tape; returned by the operating system.

length — The length of this file in small page blocks. Set by the operating system, or as indicated under the w option of this message.

wva — When the w field is zero, this field is the working virtual bit address corresponding to the virtual start address of the file.

blength, bva — If the user requests file maps to be delivered to the program buffer (map=1), the user must also set these fields to specify the length and virtual bit address where the operating system is to store the maps. If the blength field is zero, the maps are not transferred to a buffer.

lp — Field returned by the system. This field is zero for a permanent file, and 1 for a local file.

fmt — For privileged opens, this bit describes the format of the area pointed to by ptr.

0	Formatted file index entries as shown for option 1 of the LIST FILE INDEX OR SYSTEM TABLE message (f=0009)
1	Unformatted file index entries (same as described for the file index table in section 2)

ptr — For privileged opens, this field contains the virtual bit address (furnished by the user) of the first word of a 16-word area in which the operating system is to return a formatted copy of the file index table entry for the opened file. The format of the file index table entry copy is the same as for the Beta portion of the LIST FILE INDEX OR SYSTEM TABLE message (f=0009), option 1, or unformatted as described in section 2, depending on the setting of fmt. The first word of the file index table entry copy must be prefilled by the user with the user number or the pool name of the file to be opened; the second word contains the file name; and the remaining words contain the file index table, as supplied by the operating system.

mlength — Maximum length in small pages to which the file can be extended. If the file is not opened with write access, mlength = length is returned by the system. If the file is opened with write access, the system returns mlength = length + (length*EXTSIZ/100), where EXTSIZ is an installation parameter.

nc — File extendability and contiguity; returned by the operating system if the open is successful. The values are:

0	Contiguous create was not requested but extensions are allowed
1	Contiguous create was not requested and extensions are not allowed
2	Contiguous create was requested and extensions are allowed
3	Contiguous create was requested but extensions are not allowed

fsto — File segment table (FST) ordinal; returned by the operating system if the open was successful.

saddr — When the c field is 2, this field must contain the beginning sector address of the file to be opened.

unit — Same as the unit field in Beta(2).

Figure 4-4. OPEN FILE Message Format (Sheet 4 of 4)

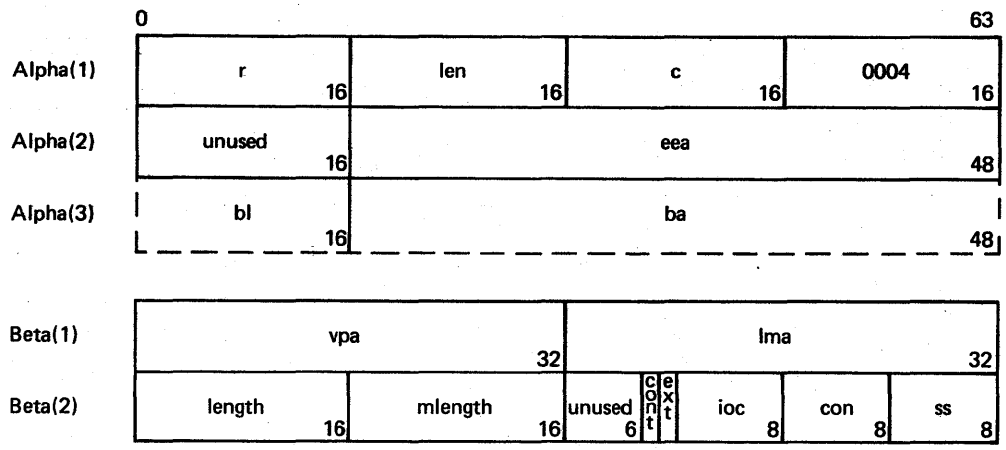
MAP (f=0004)

The MAP message gives a program access to a virtual region by defining a correspondence of virtual addresses to physical mass storage addresses. The process of defining the virtual region associated with a file is called mapping-in the file. Once a program maps in a file, the program can perform implicit reads and writes on the file. The message might also be used to release (map out) a virtual region by erasing the correspondence of the virtual addresses with mass storage. Some of the same operations that are performed using a MAP message can also be performed using an OPEN FILE or CLOSE FILE message. The mass storage space that is being mapped could contain a file already defined and opened, or it could be space that is not associated with any file (free space).

Before virtual space can be accessed implicitly, the definition of that space must be cataloged in the implicit map area of the program's minus page. The definition can be made using MAP with the map-in option. Up to 40 noncontiguous address regions can be cataloged. The user relates a virtual starting address and length with a mass storage address of an open file or free space and indicates the access rights pertaining to that virtual region. The operating system makes the necessary entries in the bound implicit map (for an open file) or drop file map (for free space) of the program. Overlaps of space are signaled as an error. If all entries of a map are full, an error is signaled and no further map-in calls are permitted until some space is released with a map-out.

The map-out option allows for release of virtual address space. Virtual address space that has been mapped out is no longer accessible to the program, but the mass storage file itself is not closed (the input/output connector for the file remains intact). The mass storage region can, after the map out, be mapped in again to the same or other virtual space. Mapping out free space causes the corresponding drop file map entries to be deleted and frees the mass storage space for reassignment. If the mass storage file represented by a virtual region has write access and is mapped out, all modified pages of that space are written on that mass storage file before the map-out process is complete. If the file itself did not have write access, all modified pages are lost through the map-out process.

The MAP call must not be used with files opened for explicit input/output or with privileged opens. Also, source files cannot be mapped in. The format of the message is shown in figure 4-5.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

- 1 — Error has been returned to the ss field of Beta(2)
- 214 — Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion.

c — Map mode:

- 0 — Map-in the file specified in the ioc field
- 1 — Map-out the file specified in the ioc field (both the bound implicit map and drop file map entries are altered)
- 2 — Map-out only this program's drop file (a c field value of zero must be used to map-in a drop file)

eea — Virtual bit address where control transfers if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

Figure 4-5. MAP Message Format (Sheet 1 of 3)

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

vpa — Virtual page address of the first small page of the space defined.

lma — Logical mass storage sector address in the file associated with the virtual page address. If this field is #FFFF, free space is appended as defined by the virtual page address and length fields.

length — Length of the virtual region in small page blocks. If this call is not for free space, the space on the mass storage file must be contiguous.

mlength — Maximum number of small pages to which this virtual region of the file can be extended. If the file was not created with write access, the system returns mlength=mlength when the map-in is successful. If the file was created with write access and the last page of the file is included in the map, the system returns mlength = length + (origlen *EXTSIZ/100) - (slen-origlen), where origlen is the original length of the file, EXTSIZ is an installation parameter, and slen is the current length of the file. If input or output is subsequently to be performed after map-in, the mlength value must be converted by the user to a binary byte count (length of the file in bytes) and stored in the index limit (IL) field of the appropriate System Record Manager file information table, using the SETFIT macro.

cont — File contiguity; set by the operating system after a successful map-in (value is set at creation time). This field is zero if the file was not created contiguously (in two segments), or set to 1 if the file was created

ext — File extendability; set by the user. The values are:

0	Extensions allowed if extensions were not prehibited on creation of the file (same as ext=2)
1	No extensions allowed (same as ext=3)

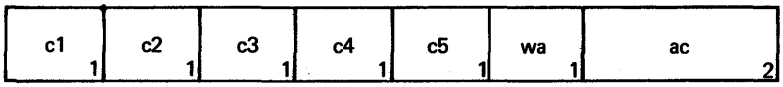
Value which was set at creation time is returned by the operating system after a successful map-in. This field is zero if extensions were allowed or set to 1 if not allowed. If the file was created with no extensions allowed, it could not have been mapped-in with extensions allowed; however, if the file was created with extensions allowed, it can be mapped-in with either extensions allowed or not allowed.

ioc — Input/output connector number for the mass storage file being mapped:

#0 to #F, #12 to #47	Mass storage file map-in or map-out
#10	Source file map-out
#11	Drop file map-in or map-out

A source file cannot be mapped in.

con — A set of eight bits providing control information, as follows:



c3=0	Small page map request
=1	Large page map request
wa=0	Get access rights (determined when the file was opened) from the input/output connector
=1	Get access rights from the ac field if the lok field from the input/output connector permits
ac=0	No read or write access
=1	Read access
=2	Write access
=3	Both read and write access

Bits c1, c2, c4, and c5 are not used. The wa and ac fields are examined by the system when mapping in files associated with input/output connectors #0 through #F.

ss — Error response field. The values are:

0	Normal completion
1	Virtual address overlap of file space
3	Length field in a map-out message is zero or greater than the length in the map
4	Sector count in the large page request is not modulo 128
5	Input/output connector does not exist or the mode specified in the input/output connector is not implicit
6	Virtual address is the same as that of an existing ADVISE call
7	Bound implicit map was full at map-in

Figure 4-5. MAP Message Format (Sheet 2 of 3)

8	Logical mass storage address plus length exceeds the file length
9	Page requested for map-out is locked in
A	Space is undefined at map-out
B	Map entry is specified to be a large page but the virtual address is not on a large page boundary
C	Bound implicit map is full at map-out
D	Input/output connector is not proper for a free space request
E	Drop file map is full at map-out
F	Not enough room in the drop file for free space map-in
10	Mass storage file index table entry cannot be found
11	Virtual address overlap of free space
12	For a map-in request, no read access was specified; map-in has not been performed
13	File is privileged-opened by the user and cannot be mapped

Figure 4-5. MAP Message Format (Sheet 3 of 3)

CLOSE FILE (f=0005)

Mass Storage Files

A program can issue the CLOSE FILE message to sever its connection to a mass storage file. After the file has been closed, the program no longer has access to the file through the severed connection, although other unsevered input/output connections might remain. Existence of the mass storage file is not affected by a close, but some file attributes in the file index table entry for the file are modified, and virtual address space associated with the file is no longer defined. A file that has been privileged created or privileged opened can be closed only with a privileged close. The user must do a privileged close before doing a privileged destroy.

When a file is closed, the operating system gives the file to an output processor if the activity count (the count of programs accessing the file, that is, of input/output connectors for the file) is zero and the management category is output. The first character of the filename indicates the particular output processor to receive the file (see Device Type Output Files in section 3). Other ways of outputting a file are to use the ROUTE AND FILE DISPOSITION message (f=000D) or the GIVE FILE message (f=0008).

When a file opened for implicit input and output and with write access is closed, modified pages of the file are rewritten in mass storage before the close function has completed. If the file does not have write access, modified pages are lost at the time the close function completes.

All outstanding input/output requests are completed before any file index table changes are made. The file index table entry will exist in its new state only at the completion of CLOSE FILE message processing.

Magnetic Tape Files

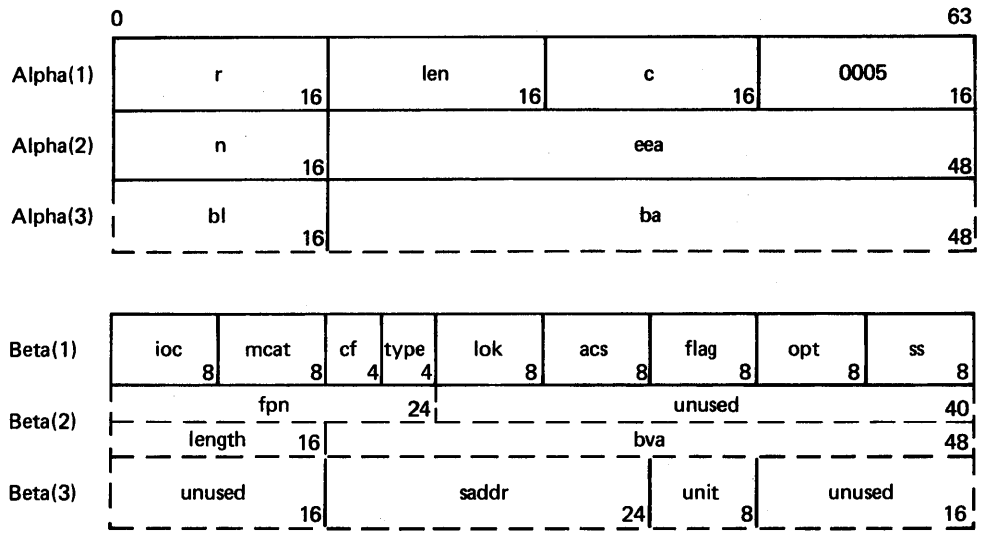
The CLOSE FILE message is also used to relinquish access to a file which resides on magnetic tape. After the file has been closed, the program no longer has access to the file even though the tape remains assigned to the job. Access to subsequent files residing on the same tape can then be obtained by this program or another program in the same batch job or interactive session by issuing an OPEN FILE message.

For a tape file, the CLOSE FILE message allows only the access attributes (acs and loc) and the current file position numbers to be modified in the file index table entry. File type cannot be changed.

The user supplies the current file position number. The system records it in the file index table entry and returns it in a subsequent OPEN FILE message.

Note that only the following Beta fields are used for a magnetic tape file: ioc, mcat, cf (bit c2), lok, acs, ss, and fpn.

This message cannot be used to close drop files or source files (input/output connectors #10 and #11). The format of the CLOSE FILE message is shown in figure 4-6. (The Beta portion of the message can actually consist of more than one of the 1-, 2-, or 3-word sets shown in the figure.)



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in an ss field of Beta(1)
2	User is not privileged (privileged closes only)
211	Number of files specified in this call is illegal (the n field is zero or greater than 16)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified
270	Beta portion is in read-only space

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. This field must be a multiple of 3 for a regular close issued by a USER1 routine (c=2); this field must be a multiple of 2 for other regular closes (c=0). For a privileged close (c=1), Beta must have a length of at least one word per file closed.

c — Close mode:

0	Regular close
1	Privileged close
2	Can be issued by USER1 routines only (system also performs return of the files)

n — Number of files closed by this message; maximum is 16.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

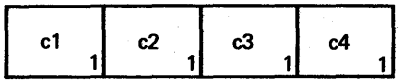
Figure 4-6. CLOSE FILE Message Format (Sheet 1 of 3)

ioc Input/output connector number of the file being closed.

mcat File management category of the file being closed; stored in the file index table if the flag field is 2. The categories are:

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
4	Magnetic tape file
5	User-created drop file
7	Batch file

cf A set of four control bits, as follows:



c1=0 Do not change the file type in the file index table
=1 Change the type in the file index table to the value given in the type field
c2=0 Do not change the file access and lockout fields in the file index table
=1 Change the file index table entry's file access field and lockout field to the values given in the acs and lok fields of this message
c3=0 Do not change the drop file size
=1 Change the drop file size in the file index table to that given in the length field
c4=0 Do not remove the drop file length from the file index table
=1 Remove the drop file length from the file index table

These flags can cause changes to be made in the file index table if the file ownership is: private; pool, and the user is the pool boss; or public, and the user is privileged. This field must be all zeros for privileged closes (the ss field is A otherwise).

type File type:

0	Physical data file
1	Virtual data file
2	Virtual code file

lok File access lockout; stored in the file index table if the c2 field is 1. The values are:

0	No access is locked out (same as lok=4)
1	Write access is locked out (same as lok=5)
2	Read access is locked out (same as lok=6)
3	Read and write access are both locked out (same as lok=7)

acs Default file access permission; stored in the file index table if the c2 field is 1. The values are:

0	No access is permitted (same as acs=4)
1	Write access is permitted (same as acs=5)
2	Read access is permitted (same as acs=6)
3	Read and write access are both permitted (same as acs=7)

flag Flag for special action. The values are:

0	Ignore the contents of Beta(2) and the mcat field
1	Change the file index table's base virtual address to that given in the bva field
2	Change the file index table's file management category to that given in the mcat field
3	Do both flag=1 and flag=2

opt Option field; for a privileged close only. If this field is zero, the file is not to be destroyed with this call; or, if it is 1, the file is to be destroyed with this call.

ss Error response code. The values are:

1	Input/output connector was not for a mass storage file
2	Input/output connector number was out-of-range
3	Attempt to alter a public file index table entry; the input/output connector is cleared, but no information is altered in the file index table (file is closed)
4	File type, access right, or lockout specified by this request is illegal (file is closed)
5	Close was requested for a file, one of whose pages is still locked in

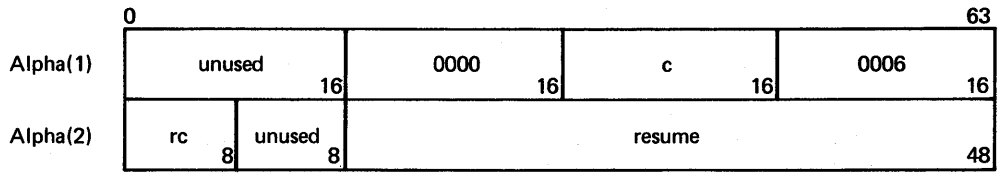
Figure 4-6. CLOSE FILE Message Format (Sheet 2 of 3)

	6	A scratch or output file is open to another program of this user (file is closed to this problem program but not destroyed or given)
	7	Invalid name for a file with a management category of output (file is closed)
	8	Specified input/output connector was not open
	9	Drop file map is full
	A	Format error (privileged close only)
	B	Nonprivileged user (privileged close only)
	C	File was not found or not attached
	D	Illegal user number for a regular close issued by a USER1 routine (c=2 only)
fpn	File position number of the last file processed on a multifile tape. Supplied by the user.	
length	Length of the drop file in small page blocks; set in the file index table if the c3 field is 1.	
bva	Base virtual bit address corresponding to the first word of the mass storage file; set in the file index table when the flag field is 1 or 3.	
saddr	Beginning sector address of the file to be closed (c=2 only).	
unit	Logical unit number of the device on which the file to be closed resides (c=2 only).	

Figure 4-6. CLOSE FILE Message Format (Sheet 3 of 3)

TERMINATE (f=0006)

A user program can issue a TERMINATE message to signal the operating system that it has temporarily completed execution. All lower level controllees are also terminated. The message consists of an Alpha portion only, as shown in figure 4-7.



c	Indicates disposition of the drop file when the program is removed from main memory. The values are:
	0 Drop file, scratch files, and output-type files are preserved (but not made permanent) so the program can be restarted. All modified pages belonging to write access files overwrite their current disk images and all other modified pages are written to the drop file. The resume address is stored in the drop file's minus page.
	1 Drop file and scratch files are destroyed. Give output-type files to the output processor.
	2 Same as option 0 except the terminate state is set to report an abort (3D).
rc	Return code; set by the user. The values are:
	0 Successful completion
	4 Nonfatal error
	8 Fatal error
	This return code value is also referred to as the termination value.
resume	Virtual bit address at which the program is to be resumed when it is restarted.

Figure 4-7. TERMINATE Message Format

LIST UNFORMATTED FILE INDEX TABLE (f=0007)

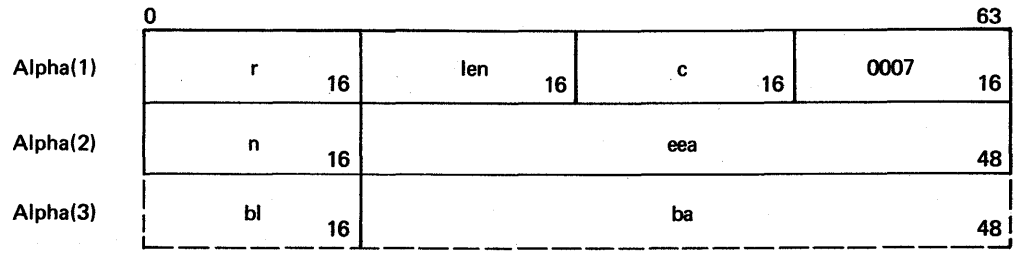
The LIST UNFORMATTED FILE INDEX TABLE system message can retrieve unformatted copies of one or more file index table entries. The message issuer can specify the file ownership category and file attributes of the entries to be returned.

The file index entries are returned in the Beta portion of the message. The Beta length must be a multiple of the file index entry length (refer to figure 2-1). Qualifiers for the file index table search are specified only in the first entry length of the Beta.

The file index table entry format is shown in figure 2-1. The message format is shown in figure 4-8.

NOTE

Because CYBER 200 OS treats magnetic tape files as local (private) files, file index entries are returned for tape files unless excluded by the message option field or Beta qualifiers.



r — Response code returned by CYBER 200 OS when message processing completes.

0	No errors.
1	No files exist which match the options and qualifiers specified.
2	More file entries exist than can fit in the Beta portion. At least one more file entry exists for the owner, but the file index entry has not been checked for matching qualifying fields.
3	Caller is not a member of a pool or the pool is not attached.
4	Illegal message option (c field).
5	Incorrect file index entry length (w field)
214	Beta length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len — Total Beta length. If len is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta buffer; otherwise, the Beta buffer is assumed to begin at Alpha(3). The length must be a multiple of the file index entry length.

w — Current file index entry length (refer to figure 2-1). If an incorrect value is specified, a response code of 5 is returned and the system returns the correct value in this field.

c — Message option specifying the file index information to be returned.

0	Public file entries.
1	Pool file entries
2	Private file entries.
3	Entries for private files attached to this suffix.
4	Entries found according to the file search hierarchy. The first search is for a private file attached to this suffix. If no file is found, the second search is for a pool file. If again, no file is found, the third search is for a public file.

n — Maximum number of file index entries to return. If fewer than n qualifying file entries are found, this field is reset to the number of file entries found.

eea — Error exit address; virtual bit address to receive control if an error occurs during message processing (r=0). If this field is zero when an error occurs, the error is considered fatal.

bl,ba — If the len value is #FFFF indicating the Alpha and Beta portions of the message are not contiguous, these fields give the length and virtual bit address of the first full word of the Beta portion.

Figure 4-8. List Unformatted File Index Table (f=0007) (Sheet 1 of 2)

The Beta format is the same as the file index table format (figure 2-1). The Beta buffer length must be at least the file index table length multiplied by the number of entries to be returned (n).

Values specified in the first entry of the Beta buffer are used as qualifiers in the file index table search. A pool name must be specified for option c=1; all other qualifiers are optional.

The following qualifiers can be used with any option (c=0,1,2,3, or 4) (refer to figure 2-1 for the field format).

name	File name.
qf	Queue flag indicating whether the file has been read by the input queue manager.
mcat	Management category.
fidc	Disposition code.
fiic	Internal format characteristic.
fiec	External format characteristic.
fisid	Source or destination processor mainframe identifier.
fizip	Destination processor zip number.
saddr	Beginning sector address of file.
slun	Logical unit number of this segment.

If within the Beta portion, a field in the first file index entry length is nonzero, the specified value is used as a qualifier. Zero cannot be used as a qualifier. Only those file index table entries with field values matching the specified qualifiers are returned.

Figure 4-8. List Unformatted File Index Table (f=0007) (Sheet 2 of 2)

GIVE FILE (f=0008)

The GIVE FILE system message transfers file ownership. The file to be given must be inactive (closed) and either a local file or an attached private or pool file. (Only the pool boss can give pool files.)

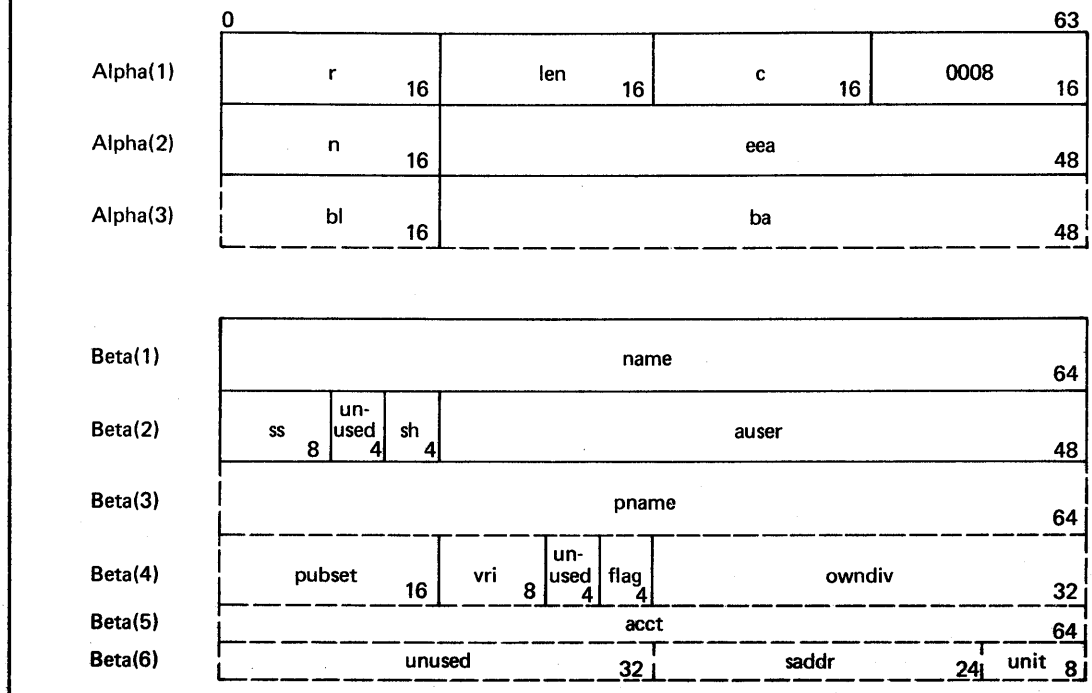
A nonprivileged user can give a file to another user or to a pool. A privileged user can also give a file to the public file list. (The file must not have the same name as an existing public file.) A USER1 routine can give a file to the input queue manager and the input queue manager can give a file to a user.

When a USER1 routine issues this message to give a file to the input queue manager, the user number and account identifier are validated and the qf field in the file index entry is set to one. When the input queue manager gives a file to a user, it must specify the sector address and logical unit number, as well as the name, of the file.

CYBER 200 OS issues the GIVE message to give files to an output processor for release to an appropriate device. Processing of print file families is deferred until the end name (PXXhmmss) is recognized (refer to Job Processing in section 3 of volume 1); other files are processed immediately. The user number of the file giver is stored in the file index table entry for the file only if the file is given to user 9999999 for output processing (PRINTOUT or PUNCHOUT routines).

The user can issue the CLOSE FILE (f=0005) or ROUTE AND FILE DISPOSITION (f=#000D) message to output a file.

The message format is shown in figure 4-9. More than one 2- to 6-word Beta portions can be specified.



r — Response code returned by CYBER 200 OS when message processing is complete.

0	No errors.
1	Error code returned in ss field of Beta(2).
2	User is not a USER1 routine (c=2 or 3 only).
211	Number of files given was zero or greater than 16.
214	Beta buffer length error. Either the first word address of Beta plus its length is greater than the maximum virtual user address, or the Beta buffer is too small for the number of requests and length specified.

len — Total Beta length. If len is #FFFF, Alpha(3) contains the length and virtual bit address of Beta; otherwise, Beta is assumed to begin at Alpha(3). A Beta portion must be specified for each file to be given. The length of each portion depends on the give mode as follows.

c=0	Two words to give a file to a user; four words to give a file to the public file list.
c=1	Three words.
c=2	Five words.
c=3	Six words.

Figure 4-9. GIVE FILE (f=0008) (Sheet 1 of 3)

c — Give mode.

0	The file is given to the private user number specified in Beta(2) or, if the specified user number is zero and the issuer is privileged, the file is given to the public file list.
1	The file is given to the pool specified in Beta(3).
2	If the issuer is a USER1 routine, the file is given to the input queue manager.
3	If the issuer is a USER1 routine, the file is given to the user number specified in Beta(2). The input queue manager uses this option.

n — Number of files to be given; maximum is 16.

eea — Error exit address; virtual bit address to receive control if an error occurs during message processing (r=0). If this field is zero when an error occurs, the task is aborted.

bl,ba — If the len value is #FFFF indicating the Alpha and Beta portions of the message are not contiguous, these fields give the length of Beta and the virtual bit address of its first full word.

name — File name (8 ASCII characters, left-justified, blank-filled).

ss — Error response code.

0	No errors; normal completion.
1	File recipient already has a permanent file with this name.
2	File has the same name as a public file.
3	The specified file is not attached.
4	The specified user number does not exist.
5	Output file is incorrectly named.
6	File to be given is still active.
7	Nonprivileged user specified the user number of the public file list.
8	Not used.
9	File recipient has a security classification less than that of the file.
A	Either the specified pool does not exist, or the giver does not have access to the pool.
B	Attempt to create new pack file index entry failed.
C	Disk is logically turned off.
D	Permanent file space limit exceeded for new owner.
E	No space available in system tables FILEI or UDMINI.
F	Issuer specified nonzero vri field, but the site does not use variable rate accounting.
10	Issuer attempted to set a variable rate index (vri field) for a data file being given to the public file list. Only a code file can have a variable rate index set.
11	Issuer attempted to give a file belonging to a pool for which he is not the pool boss.
12	File is a magnetic tape file.
13	File recipient already has the maximum number of files he can own.
14	Input queue manager user number specified.
15	Invalid account number (c=2 only).

sh — Access permission granted file recipient (public or pool file only). The file index table entry shacc field is set to the specified value. This field is not altered.

0	Read access.
1	Write access.
2	Read access.
4	No access.
8	No access.

auser — User number (8 ASCII characters, left-justified, blank-filled). For c=0, 1, or 3, auser is the user number of the file recipient. For c=2 (file given to the input queue manager), auser is the user number to which the job file belongs. If the issuer is privileged and the auser and c fields are zero and the len field is 4, the file is given to the public file list. When auser is nonzero, the len field must be 2 or 3.

pname — Pool name (8 ASCII characters, left-justified, blank-filled). This field specifies the pool to which the file is given when c=1.

pubset — Reserved for public file sets.

vri — Variable rate index set in the descriptor block (refer to Variable Rate Accounting in section 7). This field is only specified for virtual code files.

flag — Other operations to be performed when giving the file to the public file list. This field is used only when the auser and c fields are both zero.

1	Clear the originating user field in the file index table.
2	Modify the owner division field in the file index table using the value in the owndiv field of this message.

Figure 4-9. GIVE FILE (f=0008) (Sheet 2 of 3)

	3 Do both 1 and 2. 4 Set the file's privileged bit to allow the file, to issue privileged calls. (Privileged status is not extended to the file's controllees.) The default is to clear the privileged bit and not allow the file to issue privileged calls. 5 Do both 1 and 4. 6 Do both 2 and 4. 7 Do 1, 2, and 4.
owndiv	Owner division. This field must be set if the flag field is 2, 3, 6, or 7. This field must be set to zero if the owner is the originating user (default) or set to #FFFF if the owner is all users.
acct	Account number of the job file given to the input queue manager (c=2 only).
saddr	Beginning sector address of the job file (c=3 only).
unit	Logical unit number on which the job file resides (c=3 only).

Figure 4-9. GIVE FILE (f=0008) (Sheet 3 of 3)

LIST FILE INDEX OR SYSTEM TABLE (f=0009)

With this message, a user can retrieve a formatted copy of part or all of certain system tables. The Beta format for option 1 of this message is used by the privileged option of the CREATE FILE message, privileged option of the OPEN FILE message, and the GET PACK LABEL AND PFI message.

For c field values of 0, 1, 8, C, D, and E, the number of Beta words returned for each file entry is specified by the quantity, len divided by n. For example, if four files (n=4) were to be listed and the len field is 16, only the first four Beta words of information for each file would be returned. To get all 16 words of information for each file would require that n=4 and the len field be at least 64. For options 6, 7, A, and B, the n field must be 4.

The system returns only entries for mass storage files, not magnetic tape file entries. To retrieve file entries for tape, use system message 0007 (LIST UNFORMATTED FILE INDEX TABLE).

Because the operating system moves the entries in the file index table about during the execution of a job or task, the user of multiple LIST FILE INDEX OR SYSTEM TABLE requests to execute a single option cannot rely on the file name returned in the last Beta entry to indicate where scanning is to begin for the next request.

The format of the LIST FILE INDEX OR SYSTEM TABLE message is shown in figure 4-10. (For some of the message options, the Beta portion of the message can actually consist of more than one of the multiword sets shown in the figure.)

For c field values of 0, 1, 6, 7, A, B, C, D, and E, one Beta word group is set up by the user, and several Beta word groups are returned by the operating system. For a c field value of 8, the user can set up more than one Beta word group, and the operating system returns the same number of groups as the user set up. If the file was not found, the first word of that group is zeroed.

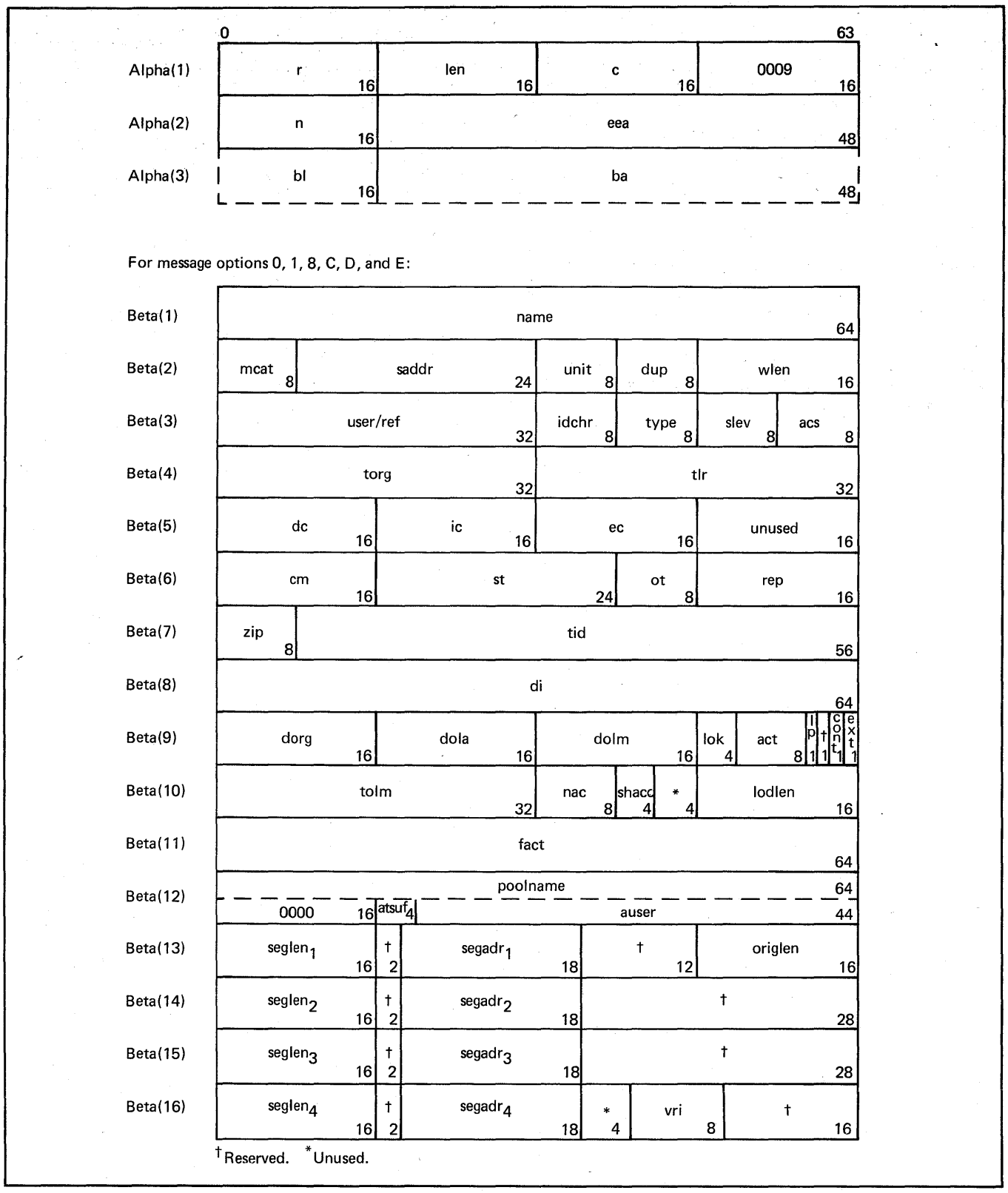
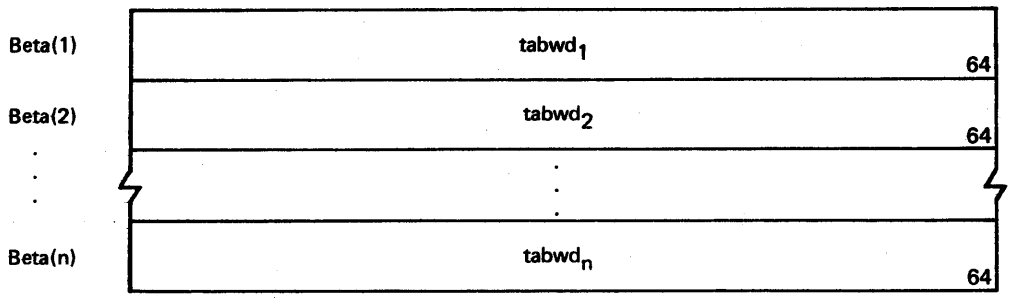
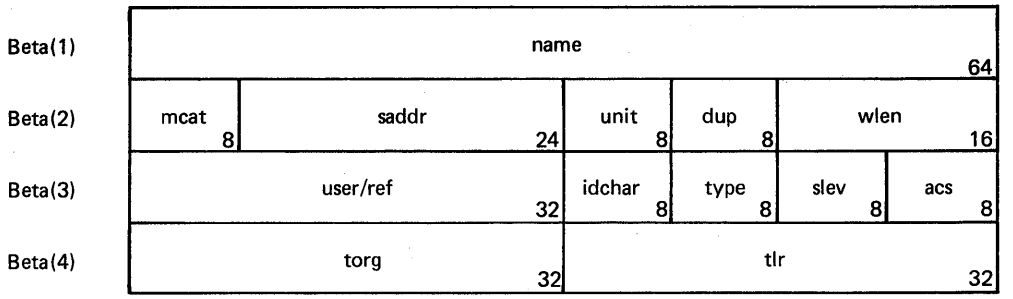


Figure 4-10. LIST FILE INDEX OR SYSTEM TABLE Message Format (Sheet 1 of 5)

For message options 2, 3, 4, 5, and 9:



For message options 6, 7, A, and B:



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Illegal message option.
2	Nonfatal; more files exist than fit in the Beta buffer. At least one more file exists for the owner but this file has not been checked for valid qualifiers (for options C, D, and E only).
3	For option E only: the caller is not a member of a pool, or a pool is not attached.
214	Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the total length in words of the Beta portion. For options 6, 7, A, and B, the len field must be a multiple of 4. For options 0, 1, 8, C, D, and E, the len field must be at least 4n. For options 2 through 5 and option 9, this field specifies the number of Beta words available for the requested system table (see the c field).

c — Message option field, specifying the system table or the file index information that the operating system is to return in the Beta portion. The values are:

0	File index table entries for all public files
1	File index table entries for all private files
2	Timecard buffer; a Beta length of at least 512 is required
3	Statistics buffer; a Beta length of at least 100 is required
4	Bank update table; a Beta length of at least 32 is required
5	Miscellaneous table; a Beta length of at least 104 is required
6	File index table entries for all batch input files (four words only are returned per file)
7	File index table entries for all private files having names beginning with the character specified in the idchar field of this message (four words only are returned per file). Beta(1) is set to binary zero if the Beta portion has not been filled; the n field is set to the number of entries returned.
8	Specified attached private file entries from the file index table. The user must supply the file names in the name fields of one or more Beta entries.
9	Disk status table; a Beta length of at least 32 is required. If more than the required number of words are specified in the bl field, the operating system resets the bl field to 32. (The system does not reset the bl field of option 2, 3, 4, or 5.)
A	File index table entries for all private files having names beginning with the characters specified in the name field (four words only are returned per file)

Figure 4-10. LIST FILE INDEX OR SYSTEM TABLE Message Format (Sheet 2 of 5)

	B	File index table entries for all public files having names beginning with the characters specified in the name field (four words only are returned per file)
	C	File index table entries for all private files that qualify; nonzero entries in the first Beta entry act as qualifiers. Valid qualifiers are name, unit, dc, ic, ec, st, and zip. If any of these fields are nonzero, only an entry with matching values for those fields is returned. The first Beta entry is overwritten with the returning entry.
	D	File index table entries for all public files that qualify; nonzero fields in the first Beta entry act as qualifiers (valid qualifiers are the same as for option C). The first Beta entry is overwritten as for option C.
	E	File index table entries for all pool files that belong to the pool named in Beta(12) and that qualify; nonzero fields in the first Beta entry act as qualifiers (valid qualifiers are the same as for option C). The pool must have previously been attached. The first Beta entry is overwritten as for option C.

n

For c field values of 0, 1, 8, C, D, and E, n is the number of files to be listed; the quantity, len divided by n, specifies the number of words returned per file index table entry. For the other options, n is the size of the table to be listed (in words) and the Beta area should be at least n words long. The operating system moves words from the table into the Beta area until either the table or the Beta area is exhausted. The value of n must always be greater than zero.

For c field values of 0, 1, 6, 7, A, B, C, D, and E, if the operating system finds fewer than n files to list, it resets n to the number of files found. For options 6, 7, A, and B, four words are returned for each file. For options 0, 1, 8, C, D, and E, the number of Beta words specified by the len field can be greater than 16 per file entry, but only the first 16 words of the len/n words reserved for the entry are used (that is, it is possible to have spare words between Beta entries).

For options 2, 3, 4, 5, and 9, n is not used (length of the Beta portion specified by the len field alone).

eea

Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba

If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion. For a c field value of 9, the operating system sets the bl field to the length of the table.

name

For option 8, the name field of each Beta entry supplied must contain the ASCII name of a file for which information is to be returned. The name must be left-justified with blank fill. The operating system returns as many Beta entries as file names supplied.

For options A and B, the user must supply one string of up to eight ASCII characters, left-justified with binary zero fill, in the name field; the operating system overwrites it with the name of a file whose name begins with that string. Entries for all files whose names begin with the string are returned.

For options C, D, and E, the name field in Beta(1) acts as a qualifier and must either be set to binary zero or else be filled with an ASCII file name, left-justified with blank fill. See also the unit, dc, ic, ec, st, and zip fields.

The name field is not used for options 0, 1, 6, and 7.

The operating system places a zero here if the name given is not found when C=7 or 8.

$tabwd_i$

One word of an n-word table; returned by the operating system when the c field is 2, 3, 4, 5, or 9.

mcat

Management category for this file (all are mass storage files):

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
5	User-created drop file
6	System-created drop file
7	Batch file

saddr

First page address of the physical segment.

unit

Logical unit number of device on which the system found the file. For options C, D, or E, this field in Beta(2) acts as a qualifier and must either be set to binary zero or else be initialized with a logical unit number.

dup

Duplicate flag. Set to #D if a duplicate file entry exists; otherwise, set to 1. A duplicate file is a pre-existing file whose name and ownership are the same as those of a new file.

wlen

File length, in small pages.

user/ref

The value of the bva/lodlen field from the file index table for options 7 or C; otherwise, the value of the ref field from the file index table.

Figure 4-10. LIST FILE INDEX OR SYSTEM TABLE Message Format (Sheet 3 of 5)

idchr — Identifying character. If the user has preset this field to nonzero when the c field is 7, only files whose first nonblank, nonnull character matches this field are listed. If no such files are found, the operating system returns zero to the name field. Not used for other options.

type — File type:

0	Physical data
1	Virtual data
2	Virtual code

slev — Security level of this file, 0 to 255.

acs — Default file access permission, taken from the file index table. The values are:

0	No access is permitted (same as acs=4)
1	Write access is permitted (same as acs=5)
2	Read access is permitted (same as acs=6)
3	Read and write access are permitted (same as acs=7)

torg — System clock time, in seconds since midnight, at which the file was originated.

tlr — System clock time, in seconds since midnight, at which the file was last opened.

dc — Disposition code, indicating how the file is to be disposed. For option C, D, or E, this field in Beta(5) acts as a qualifier and must either be set to binary zero or else be initialized with a two-character disposition code. The values for this field are:

SC	Scratch; destroyed at end-of-task (default)	P1	Reserved for CYBER 200 Link Station
PR	Print on any available line printer	P2	
PU	Punch	LR	
IN	Input for batch processing. File is unstructured with ASCII data and no carriage control characters; ic is ignored.	LS	
		LT	
PF	Permanent file (for Access Station disposition only)		

ic — Internal characteristics, indicating the format of the file. For option C, D, or E, this field in Beta(5) acts as a qualifier and must either be set to binary zero or else initialized with an internal characteristic. The values for this field are:

AS	Format in 8-bit ASCII; if the file has a disposition code of PR, the file has ANSI carriage control characters
BI	Format in binary
PA or binary 0	Format is 8-bit ASCII; if the file has a disposition code of PR, the file has ASCII carriage control characters (default)

ec — External characteristics, giving the print or punch representation of the file. For option C, D, or E, this field in Beta(5) acts as a qualifier and must either be set to binary zero (in this case to indicate that the field is not a qualifier for this call) or else be initialized with an external characteristic. The values for this field are:

26	Punch in 026 keypunch format	B4	Reserved for CYBER 200 Link Station
29 or binary 0	Punch in 029 keypunch format	B6	
		A4	
80	Punch in 80-column binary format	A6	
		A9	

Default is 29.

cm — Conversion mode, indicating the type of conversion to be done on the Access Station (st=AST):

DI	Display code (64-character set)
EC	Extended display code (128-character set)
BI	Binary

st — A three-character site identifier. For option C, D, or E, this field in Beta(6) acts as a qualifier and must either be set to binary zero or else be initialized with a site identifier. The values are:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of a CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of a CYBER 200 Link Station

For other possible values for this field, see a site analyst.

Figure 4-10. LIST FILE INDEX OR SYSTEM TABLE Message Format (Sheet 4 of 5)

ot	Origin type of a file destined for the Access Station (st=AST): B Local batch E Remote batch I Interactive
rep	Retention period of the file in days.
zip	A zip for the site identifier specified in the st field. For option C, D, or E, this field in Beta(7) acts as a qualifier and must either be set to binary zero or else be initialized with a zip. For the possible values for this field, see a site analyst.
tid	Terminal identifier. For a file destined for the Access Station, a user number of one to seven characters that identifies the user on processor st to which the file is routed. The central site is indicated by tid=0. (Not meaningful for files destined for CYBER 200.)
di	Delivery information for the Access Station; one to eight 8-bit characters held with the file.
dorg	Date this file was originated, in the format of the dorg field of the file index table.
dola	Date of the last access to this file, in the format of the dorg field of the file index table.
dolm	Date this file was last opened with write access, in the format of the dorg field of the file index table.
lok	File access lockout. The values are: 0 No access is locked out (same as lok=4) 1 Write access is locked out (same as lok=5) 2 Read access is locked out (same as lok=6) 3 Read and write access are both locked out (same as lok=7)
act	Current activity count.
lp	Field indicates if file is a local or permanent file. This field is zero for a permanent file, or 1 for a local file.
cont	File contiguity, set at file creation time. This field is zero if the file need not be created contiguously (is in two segments), or set to 1 if the file must be created contiguously.
ext	File extendability, set at the time the file was created or opened. This field is zero if extensions are allowed, or set to 1 if extensions are not allowed.
tolm	Time, in seconds since midnight, when this file was last opened with write access.
nac	Access Station area code.
shacc	Shared access permission for pool and public file sets. The values are: 0 No shared access is permitted (same as shacc=4) 1 Write shared access is permitted (same as shacc=5) 2 Read shared access is permitted (same as shacc=6) 3 Read and write shared access are both permitted (same as shacc=7)
lodlen	Length, in small pages, of the program's drop file. Meaningful only for file type 2; otherwise, this field is zero.
fact	ASCII account number, left-justified with blank fill.
poolname	For option E, a poolname in ASCII, left-justified with blank fill. Set by the user.
atsuf	For all options other than E, the suffix designator: 0 File is not attached ≠0 File is attached; atsuf contains the value (1, 2, 3, or 4) corresponding to suffix (A, B, C, or D) to which this file is attached
auser	For all options other than E, the binary user number issuing the request; returned by the operating system.
$seglen_i$	Length of the i^{th} file segment, in small pages. If $seglen_1$ is zero, the file is an operating system version 1.1.5 file, contiguous beginning at saddr, and wlen blocks long.
$segadr_i$	Address of the first small page of the i^{th} file segment.
origlen	Original file length requested during file creation.
vri	Variable rate index transferred to the db to be used in MCHARGE for variable rate CPU accounting. Valid for type code files only.

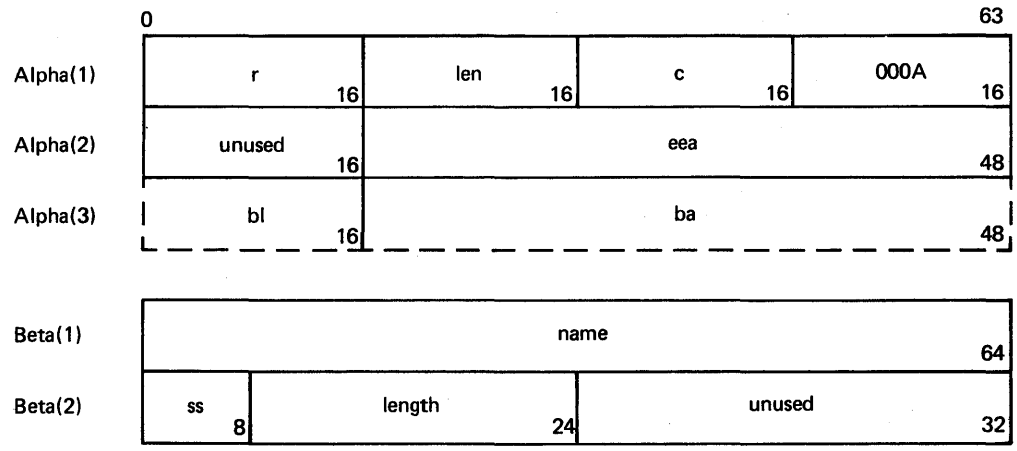
Figure 4-10. LIST FILE INDEX OR SYSTEM TABLE Message Format (Sheet 5 of 5)

REDUCE FILE LENGTH (f=000A)

A user program can issue this message to reduce the length of an existing local mass storage file or an attached permanent file. The released space is then available for the operating system to reassign it to another user. Reduction occurs at the largest absolute address end of the file. Before a file can be reduced in length, it must not be active for this or any other program in the system (that is, no input/output connectors can currently exist between this file and any program in the system). The format of the REDUCE FILE LENGTH message is shown in figure 4-11.

CHANGE FILE ATTRIBUTES (f=000B)

This message allows a user program to change the name or account number of an existing local file or an attached permanent file, to change the retention period of a file, and to change the file attributes in the file index table. A nonprivileged user can change the account number of his or her own files. Only a privileged user can change the account number of another user's files. In any case, the new account number must be valid for the user number of the owner of the file. The format of the message is shown in figure 4-12.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in the ss field of Beta(2)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion.

c — Message option field.

0	New length from length field in Beta(2).
1	New length from hba field in file index table. If file is unaccessed, the new length is one small page.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

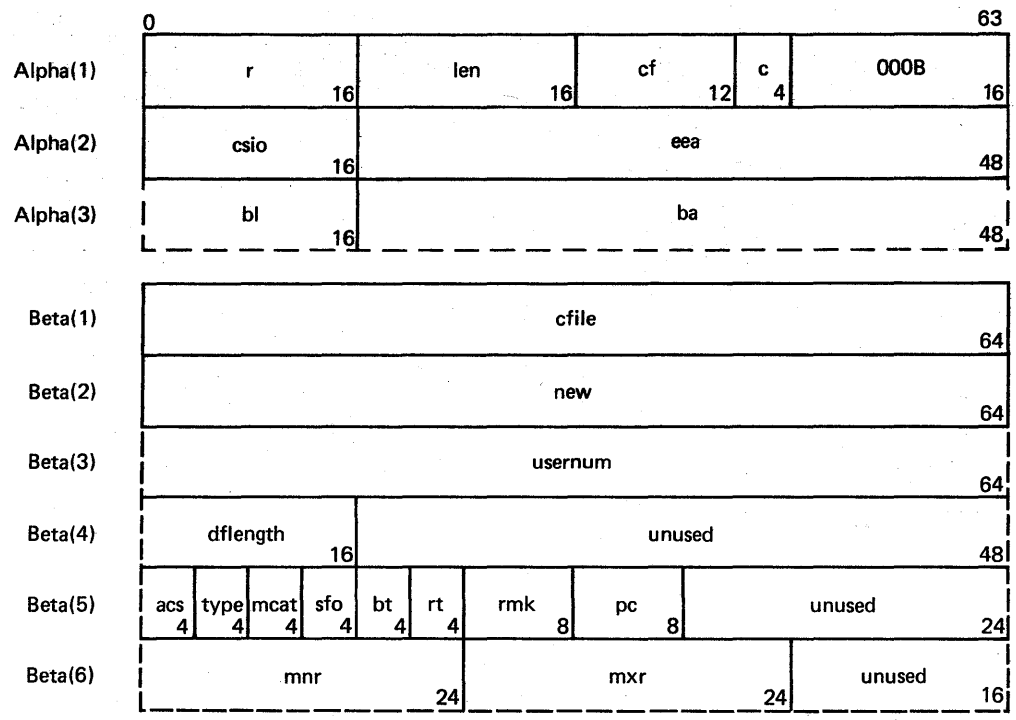
name — File name, in ASCII. File names must be in the format described in section 3.

ss — Error response field. The values:

1	New length specified for the file is greater than the existing length
2	File not attached
3	File is still open to an active program
4	Tried to reduce a virtual file to fewer than two small pages
5	Illegal option
6	Tried to reduce a magnetic tape file

length — New file length, in small pages (replaces the length parameter in the file index table). Option c=0 only

Figure 4-11. REDUCE FILE LENGTH Message Format



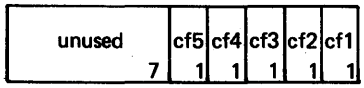
r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Current file name is still active
2	Current file name does not exist or is not attached
3	New file already exists
4	New account number is not valid
5	New file name is invalid
6	User is not privileged although the len field is 3
7	Invalid c field value
8	Duplicate permanent file name
9	Not a level-1 task (a level-1 task is a task having no controller)
10	No match found in ioc for the filename given in cfile
C	Illegal type code
D	Illegal access code
E	For virtual code file only
F	Illegal management category
11	Illegal class code
12	Illegal file organization code
13	Illegal blocking type code
14	Illegal record type code
15	Must have write access to file cfile in order to change hba
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. This field must be 2 for c=0, 2, or 3; for c=1, this field can be 2 or 3. When this field is 3, the user must be privileged. If cf is zero, len can be 2 or 3, as above. If cf is nonzero, len must be 6.

Figure 4-12. CHANGE FILE ATTRIBUTES Message Format (Sheet 1 of 4)

cf File attribute change options. Control bits are represented as follows:



cf1 File type.

0 Do not change the file type in the file index table
1 Change the type field in the file index table to the value given in the type field in Beta(5)

cf2 File access.

0 Do not change the file access in the file index table
1 Change the acs field in the file index table to the value specified in the acs field in Beta(5)

cf3 Drop file size.

0 Do not change the drop file size
1 Change the drop file size in the file index table to the value specified in the dflength field in Beta(4)

cf4 Management category.

0 Do not change the mcat field in the file index table
1 Change the mcat field in the file index table to the value specified in the mcat field in Beta(5)

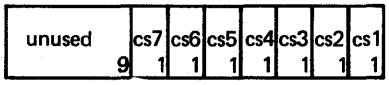
cf5 Byte address (hba).

0 Do not reset hba
1 Reset hba to 1 in the file index table and to 0 in the ioc
Note: Source file (ioc number 16) and drop file (ioc number 17) cannot be changed

c Type of change. The values are:

0 Change file attributes
1 Change the account number of the file. If a nonprivileged user selects this option, the cfile field must contain the name of one of the user's own files. If a privileged user selects this option, the cfile field may contain the name of another user's file so long as a valid number is given in Beta(3); otherwise, the cfile field must name one of the user's own files.
2 Change the account number of this executing level-1 task (a task such as a batch processor); cfile is the drop file name of the task
3 Change the retention period of this file

csio SIL file attribute change options. Control bits are represented as follows:



cs1 File organization.

0 Do not change the file organization
1 Change the file organization to that specified in the sfo field of Beta(5)

cs2 Blocking type.

0 Do not change the blocking type
1 Change the blocking type to that specified in the bt field of Beta(5)

cs3 Minimum record length.

0 Do not change the minimum record length
1 Change the minimum record length to that specified in the mnr field of Beta(6)

cs4 Maximum record length.

0 Do not change the maximum record length
1 Change the maximum record length to that specified in the mxr field of Beta(6)

cs5 Record type.

0 Do not change the record type
1 Change the record type to that specified in the rt field of Beta(5)

Figure 4-12. CHANGE FILE ATTRIBUTES Message Format (Sheet 2 of 4)

cs6 Padding character.

0	Do not change the padding character.
1	Change the padding character to that specified in the pc field of Beta(5).

cs7 Record mark.

0	Do not change the record mark
1	Change the record mark to that specified in the rmk field of Beta(5).

eea Virtual bit address to receive control if an error occurs during processing of this message (r=0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

cfile For c=0, c=1, or c=3, this field must contain the current file name in ASCII, left-justified with blank fill. If changing only the filename, the c field must be zero and new must contain the new filename (see following description of new). The cf and csio fields must also be zero. To change file attributes (cf, csio) without changing the filename, the c field and the new filename field must both be zero. The current filename (cfile) must also be given. File names must be in the format described in section 3. For c=2, this field must contain the task's drop file name.

new When the c field is zero, this field must contain the new file name in ASCII, left-justified with blank fill. For c=1 or c=2, this field must contain the new account number in up to eight ASCII characters, left-justified with blank fill. For c=1 and len=2, this field must be valid under the user number of the task issuing this message; for c=1 and len=3, this field must be valid under the user number in Beta(3). For c=3, the right most 10 bits of this field contain the new retention period in days, in binary.

usernum Binary user number under which the new account is valid (a privileged user issued a call with the c field set to 1). If the cf field is used (nonzero) and the c field options are not to be used, the c field and this field are set to zero.

dflength Length of the drop file in small page blocks when cf3 is set to 1

acs Access code when cf2 is set to 1.

0	No access is permitted
1	Write access is permitted
2	Read access is permitted
3	Read and write access is permitted

type File type code when cf1 is set to 1.

0	Physical data file
2	Virtual code file

mcat File management category when cf4 is set to 1.

0	Mass storage file
1	Scratch file
2	Output file
3	Write-temporary file
5	User-created drop file
7	Batch file

sfo File organization when cs1 is set to 1.

0	Sequential

bt Blocking type when cs2 is set to 1.

0	Not an SIL file
2	C type blocking

rt Record type when cs5 is set to 1.

0	Control word (W)
1	ANSI fixed length (F)
2	Record mark (R)
7	Undefined (U)

Figure 4-12. CHANGE FILE ATTRIBUTES Message Format (Sheet 3 of 4)

rmk	Record mark when cs7 is set to 1 (any 8-bit ASCII character).
pc	Padding character when cs6 is set to 1 (any 8-bit ASCII character).
mnr	Minimum record length when cs3 is set to 1 (24-bit user-supplied length in number of bytes).
mxr	Maximum record length when cs4 is set to 1 (24-bit user-supplied length in number of bytes).

Figure 4-12. CHANGE FILE ATTRIBUTES Message Format (Sheet 4 of 4)

GIVE TAPE ACCESS TO CONTROLLEE (f=000C)

This message allows one user program to enable (and prohibit) another user program (specifically, the first program's controllee) to access specified tape files through the first program's input/output connectors. The controllee establishes its own input/output connector for the action, but is limited by the controller in how the tape can be affected. The controllee gains access to the tape file through the same input/output connector number as the controller uses; the controllee's input/output connector does not, however, contain the tape file name. This prevents the controllee from being able to destroy the tape file.

The format of this message is shown in figure 4-13. (The Beta portion of this message can actually consist of more than one of the 2-word entries shown in the figure.)

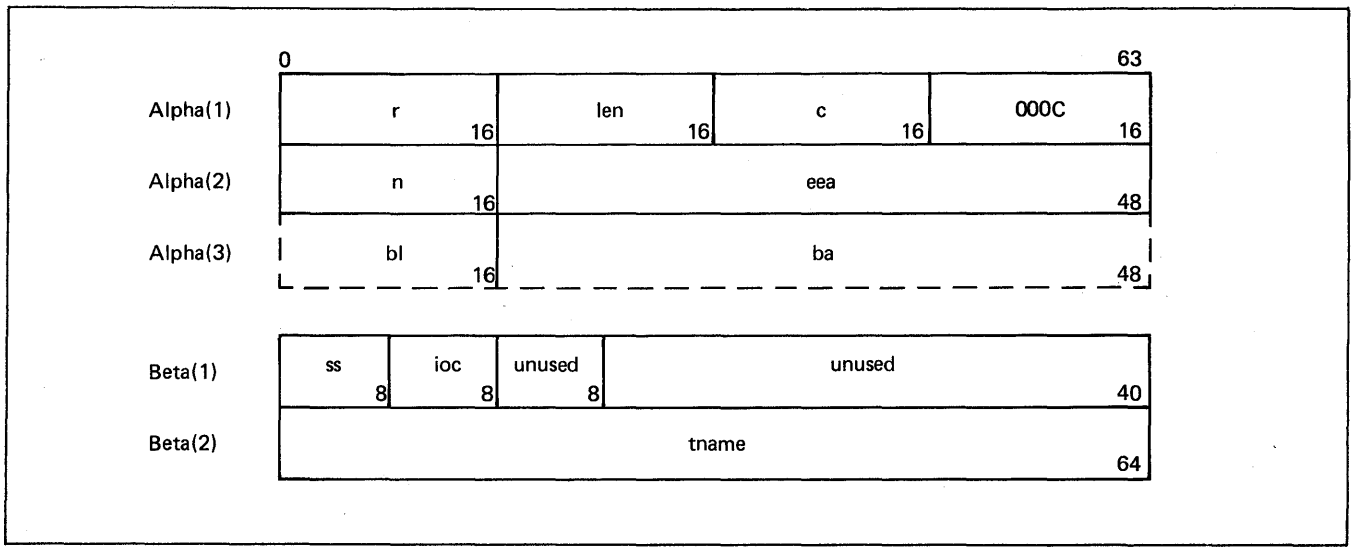


Figure 4-13. GIVE TAPE ACCESS TO CONTROLLEE Message Format (Sheet 1 of 2)

r	Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise: 1 — Error code was returned in an ss field of Beta(1) 211 — Number of tapes given by this message was illegal (the n field is zero or greater than 16) 214 — Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified
len	If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion (a multiple of 2).
c	Tape access control. This field must be zero to bestow access to the tape file named in the tname field, and must be 1 to no longer allow access to that tape file.
n	Number of tape files that the controllee can access as a result of this message; maximum is 16.
eea	Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.
ss	Error response field. The values are: 0 — Normal completion 1 — No tape name was specified 2 — Wrong input/output connector was indicated or the input/output connector is not in the range of #0 to #F 3 — Controllee already owns a private tape with the specified input/output connector number 4 — No controllee exists 6 — Controller does not own a tape by the specified name 7 — Controllee input/output connector is already in use
ioc	Input/output connector number for the tape file named in the tname field.
tname	ASCII name of one of the user's tape files, left-justified with blank fill. File names must be in the format described in section 3.

Figure 4-13. GIVE TAPE ACCESS TO CONTROLLEE Message Format (Sheet 2 of 2)

ROUTE AND FILE DISPOSITION (f=000D)

A user program can issue this message to specify the destination and disposition of a file, freeing the user from the burden of using naming conventions to accomplish routing and disposition of a file. The file to be routed must be either a local mass storage file or an attached permanent file. The file becomes an unattached permanent file at its destination. The file is released immediately unless the def field has been set to defer routing until a later time. Other ways to route a file are through the CLOSE FILE and GIVE FILE messages.

The format of the ROUTE AND FILE DISPOSITION message is shown in figure 4-14. (The Beta portion of this message can actually consist of more than one of the 7-word entries shown in the figure.)

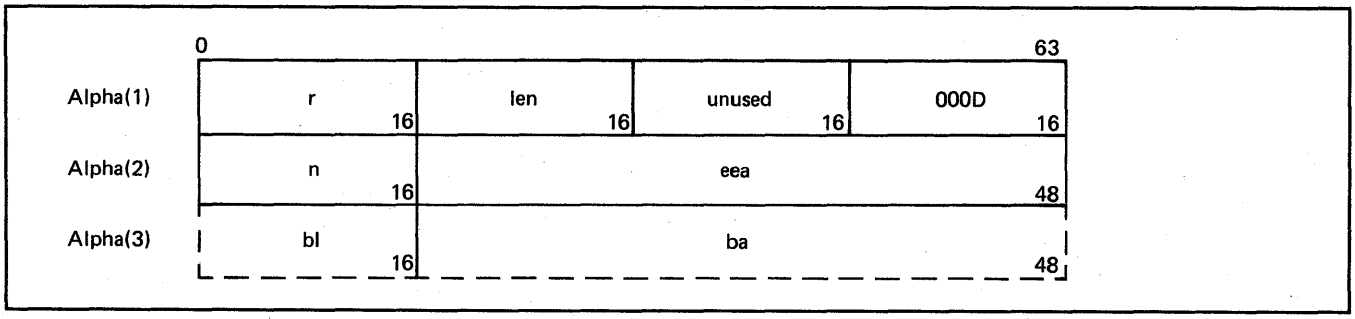
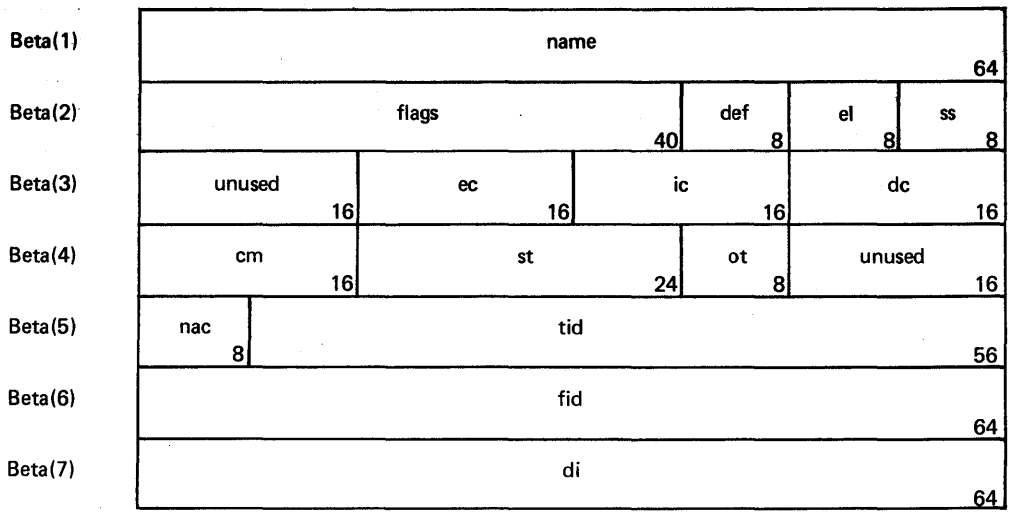


Figure 4-14. ROUTE AND FILE DISPOSITION Message Format (Sheet 1 of 3)



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Error code was returned in an ss field of Beta(2)
211	Number of files routed by this message was illegal (the n field is zero)
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion (a multiple of 7).

n — Number of files to be routed by this message. Maximum is #FFFF files.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

name — File name, in ASCII. File names must be in the format described in section 3.

flags — Flag bits. Each bit must be set to zero if the specified parameter is to be ignored, or set to 1 if the specified parameter is to be processed (the value is to be placed in the appropriate file index table entry field).

Beta(2) bits and corresponding parameters in this field, from left to right starting from zero, are:

0	Unused	4	cm	8	tid
1	ec	5	st	9	fid
2	ic	6	ot	10	di
3	dc	7	Unused	11	nac

def — If set to 1, indicates that file disposition is to be deferred. The operating system stores the information about the file into the file index table and disposes of it as requested when the file is released. Files are released by a GIVE FILE message, CLOSE FILE message, end-of-task processing, or by a ROUTE message that specifies immediate release (def=0). Routing of files to the input queue cannot be deferred. With deferred routing, the user can redefine the same file with subsequent ROUTE calls.

el — Beta entry length; must be at least 2 and no more than 7.

ss — Error response field. The values are:

1	Immediate release (def=0) of an active file
2	Immediate release (def=0) of a nonallocated file

Figure 4-14. ROUTE AND FILE DISPOSITION Message Format (Sheet 2 of 3)

3	Beta entry length (el) error
4	File must be attached before the route message is executed
5	Immediate release (def=0) with no disposition set
6	Could not write a pack file index entry
7	Illegal disposition code
8	Illegal site identifier, or the mainframe site identifier is not logged in
9	Illegal file name
A	File too big for service station (for st=ast only)
B	Attempt to route a magnetic tape file

ec — External characteristics, giving the punch representation of the file. See figure 4-10 for the possible values in this field.

ic — Internal characteristic, indicating the format of the file. See figure 4-10 for the possible values in this field.

dc — Disposition code, indicating how the file is to be disposed. See figure 4-10 for the possible values in this field.

cm — Conversion mode, indicating the type of conversion to be performed on the file when it reaches the Access Station (st=AST):

DI	Display code (64-character set)
EC	Extended display code (128-character set)
BI	Binary

st — Site identifier, identifying the processor responsible for processing the file. If the disposition code is IN (input for batch processing), this field identifies the processor on which the file is to be executed. If the disposition code specifies an output queue, this field identifies the processor on which the file is output. Current valid site identifiers are:

AST	Access Station
ADA	Reserved for the CYBER front-end portion of a CYBER 200 Link Station
ADZ or ADX	Reserved for the CYBER 200 portion of a CYBER 200 Link Station

For other possible values for this field, see a site analyst.

ot — Origin type of a file destined for the Access Station (st=AST):

B	Local batch
E	Remote batch
I	Interactive

nac — Access Station area code.

tid — Terminal identifier. For a file destined for the Access Station, a user number of one to seven characters that identifies the user on processor st to which the file is routed. The central site is indicated by tid=0. (Not meaningful for files destined for CYBER 200.) For CYBER 200 Link Station, a two-character terminal id, left-justified in the field.

fid — The first five characters of the file name that is to designate the file while it is in the output queue. Any combination of one to five letters and numbers can be specified, with the first character a letter. Two unique job sequence characters added by the system to the job name are used as the sixth and seventh characters of the file name. The eighth character (CYBER 200 only) is a blank.

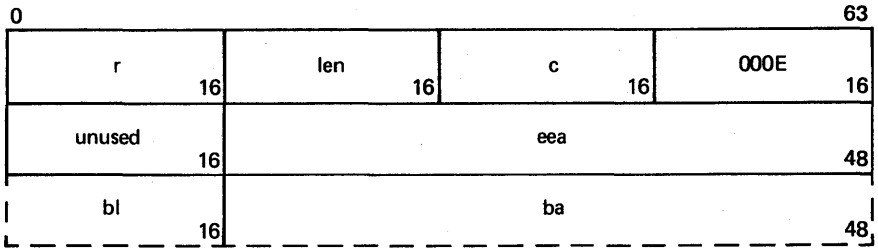
di — Delivery information for the Access Station; eight 8-bit characters (only the display code 64-character set is valid). For the CYBER 200 Link Station, eight 8-bit characters used for the banner page.

Figure 4-14. ROUTE AND FILE DISPOSITION Message Format (Sheet 3 of 3)

USER/ACCOUNTING COMMUNICATION (f=000E)

A user program can issue the USER/ACCOUNTING COMMUNICATION message to retrieve accounting statistics from the cumulative accounting buffer. This call is used by the batch processor to communicate with the accounting system. Only the accounting statistics for the program issuing the message are available to the program; the statistics are available via this message only for the duration of the job.

The format of this message is shown in figure 4-15. The Beta portion of the message is described under the c field definition. The value in each of the Beta words, except for the leftmost 16 bits of Beta(15), is a sum over all accounting periods for the task up until issuance of the USER/ACCOUNTING COMMUNICATION message.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

2	Illegal message option
3	Nonprivileged task attempted to use a privileged option; or, option 1 or 2 was issued by a user other than the public batch processor
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. For message option 4, len must be zero; for options 1 and 2, len must be 1; and for option 3, len must be 20.

c — Message option field:

0	Reserved
1	Start of this batch job; can be issued only by the public batch processor.
2	End of this batch job; can be issued only by the public batch processor.
3	Retrieve accounting information for this task or for all level controllees executed since the start of this batch job.
4	Dump accounting temporary storage to permanent storage and terminate the accounting file; can be issued only by a privileged task. No Beta portion is used for this option.

For options 1 and 2, the Beta portion of the message consists of one word, a job name of up to eight ASCII characters starting with a letter, left-justified with blank fill. For option 3, CYBER 200 returns the following Beta words:

Beta(1)	User execution CPU time, in microseconds
Beta(2)	Memory usage; at the end of each accounting period, (current working set size)*(user CPU time for current accounting period) is computed and added to a running total kept in this field
Beta(3)	Number of 16-bit bytes transferred to or from tape files
Beta(4)	Number of tape accesses (input/output requests issued) for reads and writes
Beta(5)	Number of nonread and nonwrite tape functions, such as read hardware status
Beta(6)	Virtual and resident CPU time, in microseconds, for user program execution
Beta(7)	Number of disk accesses (input/output requests issued) for large page explicit reads and writes
Beta(8)	Number of disk accesses (output requests issued) for large page implicit writes
Beta(9)	Number of disk accesses (input/output requests issued) for small page explicit reads and writes
Beta(10)	Number of disk accesses (output requests issued) for small page implicit writes
Beta(11)	Number of disk sectors transferred for explicit reads and writes
Beta(12)	Number of disk sectors transferred for implicit writes
Beta(13)	Number of disk accesses (input requests issued) that resulted from large page faults (large page implicit reads)
Beta(14)	Number of disk accesses (input requests issued) that resulted from small page faults (small page implicit reads)
Beta(15)	Current working set size (leftmost 16 bits), and the number of virtual system user calls made (rightmost 48 bits)
Beta(16)	System time units (cumulative TCHARGE calculations, integer).
Beta(17)	System billing units (cumulative MCHARGE calculations, real).
Beta(18)	Number of large pages lost.
Beta(19)	Number of small pages lost.
Beta(20)	Cumulative amount of CPU time that this task's working set size limit appeared to be too small.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-15. USER/ACCOUNTING INFORMATION Message Format

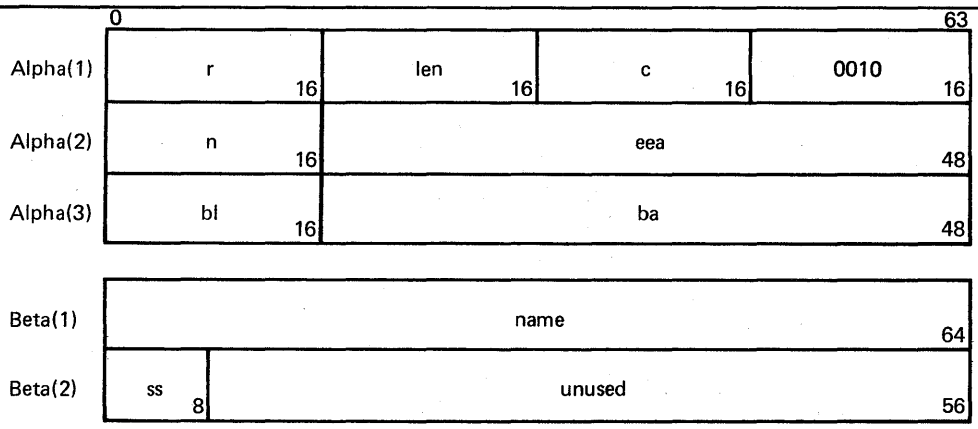
ATTACH PERMANENT FILE (f=0010)

A program issues this message to attach an existing permanent file. More than one Beta entry can be used. The format of this message is shown in figure 4-16.

GET PACK LABEL AND PFI (f=0011)

A privileged user issues this message to retrieve the pack label and pack file index (PFI) for a specified pack.

In the first call, the user initializes the Alpha words (n set to zero) and the packid field in Beta(1). The length of the Beta portion must be at least 528 (the size of a small page plus 16 words for the pack label). The pack label is returned in Beta(1) through Beta(11) and the PFI entries are returned starting at Beta(17). A count is returned to the n field indicating the number of entries plus one that have been returned in the current call (or series of calls, if more than one call is issued). If a 1 is returned in the r field of Alpha(1), more PFI entries exist and the call must be reissued to get the rest of the entries. In the second and any subsequent calls, the n field must contain the count returned in the previous call. PFI entries are then returned starting at Beta(1).



r — Response code returned by the operating system when this message has been processed. If no error occurs, the code is zero; otherwise:

- 1 — Refer to Beta field ss for the specific error. All Beta entries are processed.
- 2 — Unable to attach all files (c=1 only). A local file already exists with the same name as a permanent file, or a permanent file is attached to another suffix.
- 3 — Illegal option. Control field contains an illegal value (not 0 or 1).
- 211 — Number of files given was zero or greater than 16.
- 214 — Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len — If this field is #FFFF, Alpha(3) exists and contain the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to be located beginning at Alpha(3), and len is the length of the Beta portion of the message.

c — Control field. If this field is zero, attach files given in the Beta field. If this field is 1, attach all files (no Beta required).

n — Number of requests in this message (maximum is 16). Not used for c=1.

eea — Virtual bit address to receive control if an error occurs while this message is processed (r not equal to 0); if eea is 0 when an error occurs, the error is considered fatal.

bl, ba — If the Beta portion of the message is not contiguous to the Alpha portion (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

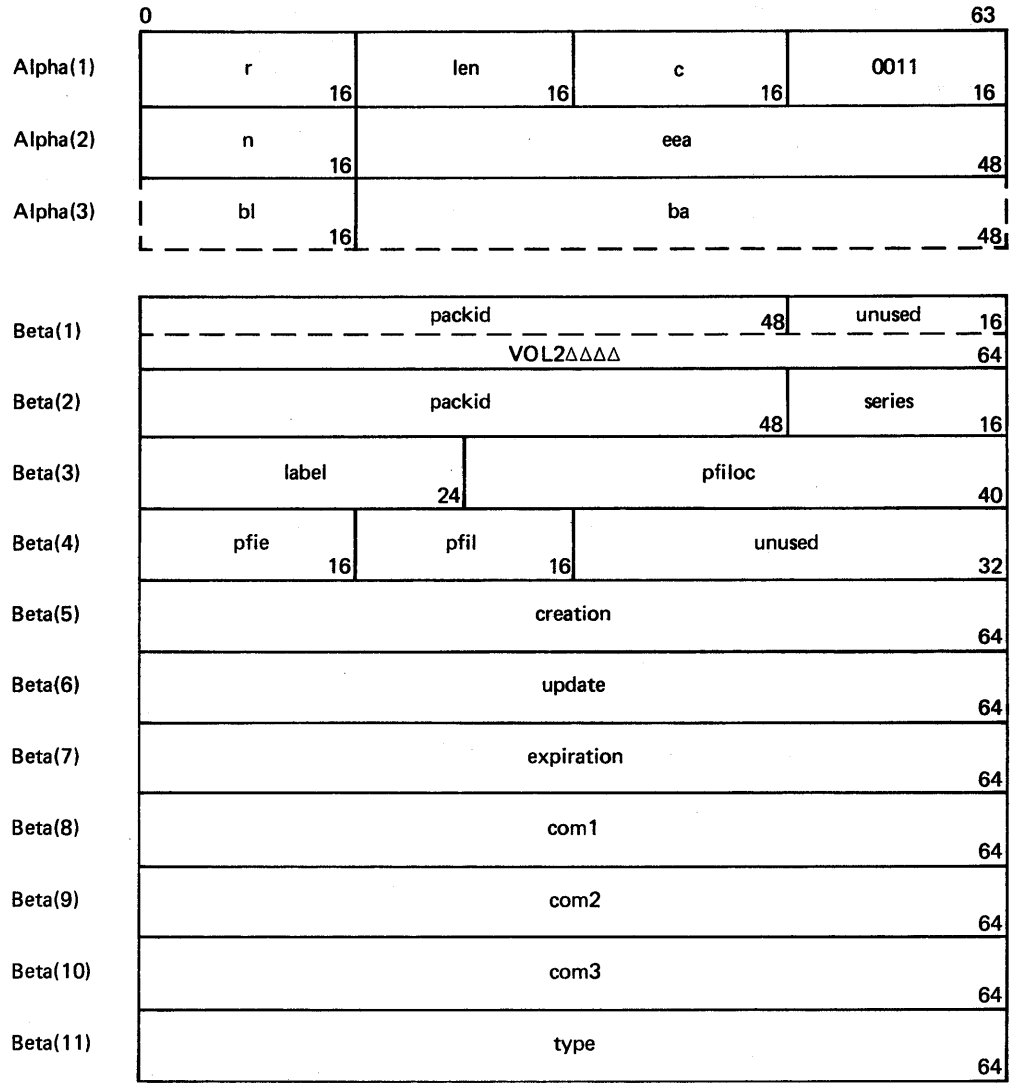
name — Name of the file in ASCII; file names (left-justified with blank fill) must be in the format described in section 3. Drop file can also be attached.

ss — Error response field:

- 0 — Normal completion
- 1 — Permanent file name not found
- 2 — File already attached to this suffix
- 3 — File attached to another suffix
- 4 — File already attached as a local file
- 5 — File is privileged open

Figure 4-16. ATTACH PERMANENT FILE Message Format

The label format, as set by the system routine NAMEPACK, is shown in figure 4-17. The format of the entries is the same as for the LIST FILE INDEX OR SYSTEM TABLE message option 1 (shown in figure 4-10), except that the user/ref field is always ref.



Beta(12) through Beta(16) are unused in the label call part only.

r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Nonfatal error; more files exist than the Beta portion could hold; reissue the call to get the rest
2	User is not privileged
3	Disk is not logically up
4	Pack identifier was not found
5	Illegal option

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. This field must be a multiple of 16, and a minimum of 528.

Figure 4-17. GET PACK LABEL AND PFI Message Format (Sheet 1 of 2)

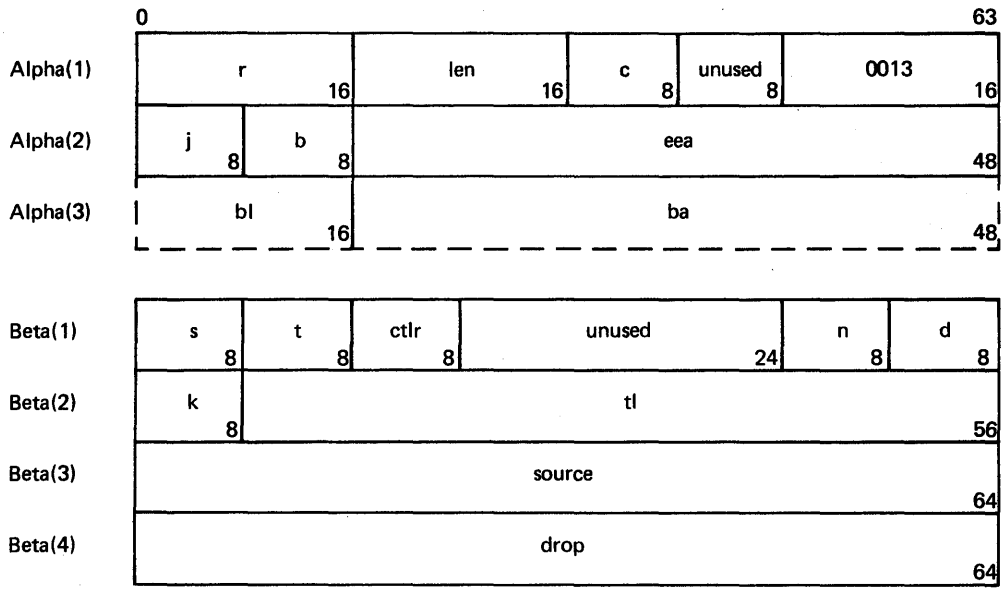
c	Message option field: 0 Return formatted PFI entries (same format as for option 1 of the LIST FILE INDEX OR SYSTEM TABLE message, f=0009) 1 Return unformatted PFI entries (same as described for the file index table in section 2)
n	This field must be set to zero by the user for the first call, causing the pack label alone to be returned. For a reissued call, this field must be set to the value that the operating system returned to the user in this field for the previous call.
eea	Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.
packid	Pack identifier of the pack sought, in ASCII, left-justified with blank fill. For a first call (n=0), the pack identifier in Beta(1) is overwritten with the returned label. When this call is reissued, the pack identifier must be reentered in Beta(1). Sixteen words of the pack label are returned (last five words are not used); all 16 words of each used pack file index entry are returned. Pack identifiers are obtained using option 9 of the LIST FILE INDEX (f=0009) message.
series	2031 in hexadecimal.
label	Disk block address of this label.
pfiloc	Disk block address of the first block of the pack file index.
pfie	Entry number of this entry within the pack file index, counting from zero.
pfil	Original length of the pack file index.
creation	ASCII date, in the format mm.dd.yy, of the creation of this label.
update	ASCII date, in the format mm.dd.yy, of the last update of the disk.
expiration	ASCII date, in the format mm.dd.yy, of the expiration of the disk.
com1, com2, com3	Three words describing the machine, serial number, and other information set by the NAMEPACK system routine.
type	841, 844, or 819 in hexadecimal; used by the operating system to determine the length of the disk pack.
	For c=0 Beta(1) through Beta(16), as returned for the pack file index, are in the same format as shown for the LIST FILE INDEX (f=0009) message, except for the user/ref field which is always ref. For c=1, they are in the same format as the file index table, described in section 2.

Figure 4-17. GET PACK LABEL AND PFI Message Format (Sheet 2 of 2)

LIST CONTROLLEE CHAIN (f=0013)

A user program can obtain a list of the controllee chain, including the program level and descriptor block number, the executable source file name, drop file name, and so forth, of each task in the chain, by using the LIST CONTROLLEE CHAIN message shown in figure 4-18. The issuing program can determine its own position in the chain by comparing fields j and b in Alpha(2) with fields s and t in Beta(1).

A total of nine levels is the maximum; that is, eight controllees with the level 1 batch processor or virtual system interactive processor. The descriptor block number is unique and is associated with the program until it terminates.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, control proceeds normally and this field contains the number of words returned in Beta; otherwise, the response code is:

1	The length specified for Beta is zero
2	An illegal option was selected
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. For message options 1, 2, and 3, len must be 4. For message option 0, len must be a multiple of 4, up to a limit of 36.

c — Message options:

0	List all controllees in the chain; controllees are listed in ascending order, starting with the job control processor
1	List only the program that issued the message
2	List only the controller of the program that issued the message
3	List only the controllee of the program that issued the message

j — Level in the controllee chain of the program that issued the message. Level numbers in this field range from 1 to 9.

b — Descriptor block number of the program that issued the message.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

s — The level in the controllee chain of the program whose name is in Beta(3). Level numbers in this field range from 2 to 9.

t — Descriptor block number of the program whose name is in Beta(3).

ctlr — Descriptor block number of the controller of the program whose name is in Beta(3):

FF	Controller is interactive processor
1	Controller is BATCHPR0

Figure 4-18. LIST CONTROLLEE CHAIN Message Format (Sheet 1 of 2)

n	Descriptor block number of the controllee (if one exists) of the program whose name is in Beta(3). Beta(3) contains the name of a program that has informed the operating system to direct messages from the controller to the n field task (the controllee) if they are not specifically directed to the controllee. The operating system places the descriptor block number of n, a program in the chain, in this field. The value of n is zero if no program has informed the operating system to direct messages to it.
d	Beta(3) contains the name of a program that has informed the operating system to direct messages from the controllee to the d field task if they are not specifically directed to the controller. The operating system places the descriptor block number of n, a program in the chain, in this field. The number is zero if the n field is zero.
k	Descriptor block number of this program's controllee. This number can be zero.
tl	Time limit of the program whose name is in Beta(3).
source	Name of the executable source file, in ASCII.
drop	Name of the drop file, in ASCII.

Figure 4-18. LIST CONTROLLEE CHAIN Message Format (Sheet 2 of 2)

SEND A MESSAGE TO CONTROLLER (f=0014)

This message is used by a program to send a string of binary or ASCII data to a program controller or the job control processor (the batch processor or virtual system interactive processor). When this message is issued, the operating system copies the data string from the Beta portion of the message into a system buffer.

When output requests are being sent to a virtual system interactive processor (a controller) from a logged-in user (its controllee) and the wait or replace option (m=0) has been selected, the system message buffer can hold up to five data strings or 4096 character bytes, whichever limit is reached first. For a logged-out user, only one data string can be held in the buffer. The data is grouped in blocks of 151 character bytes and sent, one block at a time, from the virtual system interactive processor to the output device. If the last block is fewer than 151 character bytes, an end-of-message character is added after the last character byte. The issuer of the message is responsible for formatting any multiline strings to be sent to a terminal by inserting line feed and carriage return characters at the appropriate places in the string.

If a data string from a controller is waiting for the controllee when the controllee issues this message, the data string from the controller to the controllee is lost. The controllee should check, therefore, to see if any data strings are waiting to be received before it issues this message.

If the controller is the batch processor, the message is put in the job dayfile. The format of this message is shown in figure 4-19. The Beta portion contains the string of binary or ASCII data sent to the program controller or job control processor. The maximum length of the Beta portion, when present, is 4096 character bytes. When a data string is sent in this way to the virtual system interactive processor, the processor sends it to an output device (a terminal).

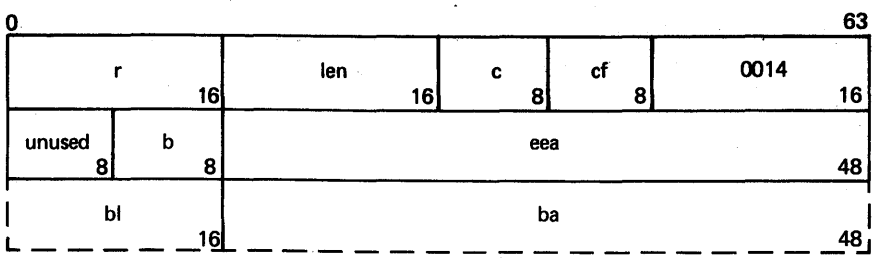
SEND A MESSAGE TO CONTROLLEE (f=0015)

A program starts a controllee running by issuing a SEND A MESSAGE TO CONTROLLEE message. The optional Beta portion of this message contains a string of binary or ASCII data for the controllee to receive as soon as the controllee has been started running. If the Beta portion is present when the operating system processes the SEND A MESSAGE TO CONTROLLEE message, the operating system copies the data string from the Beta portion into a system buffer before it starts the controllee. The controllee will have to issue a GET A MESSAGE FROM CONTROLLER OR OPERATOR message to retrieve the data string from the system buffer.

A problem arises if any controllee (except for the immediate controllee of a level-1 task) issues a GET A MESSAGE FROM CONTROLLER message when no data string is waiting in the system buffer. In this case, the controllee stops running and waits for a data string from its controller, and the next higher controller in the controllee chain is started running.

Once a controllee of the virtual system interactive processor is running, the controllee can be stopped by an ASCII character string from the terminal. The signal from the terminal stops the interactive processor's controllee and starts the interactive processor.

The format of this message is shown in figure 4-20. The maximum length of the Beta portion, when present, is 4096 character bytes.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Length of Beta in character bytes is either zero or greater than 4096
2	Illegal option was selected for this message
3	No controller matches the value of b≠0, or this task is a level-1 task; therefore, sending this particular message was pointless
4	If the notify option was selected (c=1), the controller designated was a job control processor for a logged-out user
5	If the notify option was selected, the controller designated was a job control processor for a user that has subsequently logged in under a different suffix
6	If the notify option was selected, the system output buffer is full
7	Error in sending message to job dayfile
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length in character bytes and the virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in character bytes of the Beta portion.

c — Message options:

0	If the controller to whom the data string was sent is a logged-out user, replace any existing string waiting in the buffer with the new string. If the job control processor buffer is full, stop running this program until the buffer is free.
1	If the data string cannot be sent to the controller, return control to the error exit address.
2	If the data string cannot be sent to the controller, stop running this program until the message can be sent.

cf — Control field. The values are:

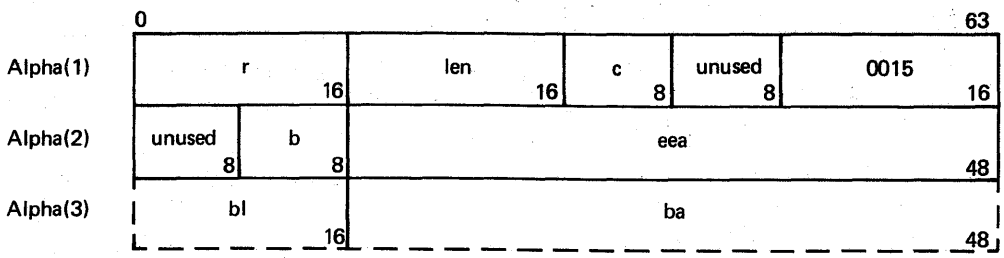
0	Send the data string to the controller. If the controller is a virtual system interactive processor or batch processor, continue running this controllee program (the program issuing this message); otherwise, start running the controller and stop running this controllee program.
2	Send the data string to the level-1 task. If the task is a virtual system interactive processor or batch processor, continue running this controllee program; otherwise, stop running this controllee program (the controller is the batch processor).

b — Descriptor block number of the controller. If the data string is to be sent directly to a level-1 task (c=2), or if this program's controller is a level-1 task, this field is ignored. If this field is zero, the data string is sent to the next higher controller.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in character bytes and virtual bit address of the first full word of the Beta portion.

Figure 4-19. SEND A MESSAGE TO CONTROLLER Message Format



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Length of Beta in character bytes is zero or greater than 4096
2	Illegal option was selected for this message
3	No controllee matches the value of the b field (for b≠0)
4	For b=0, no controllee exists
7	Error exit has been taken, since the controllee already has a text string from the controller
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length in character bytes and the virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in character bytes of the Beta portion.

c — Message options:

0	This message has a Beta portion containing a data string for the controllee. If the controllee already has a data string waiting from the controller, replace it with the new data string.
1	This message has a Beta portion containing a data string for the controllee. If the controllee already has a data string waiting from the controller, return control to the error exit address.
2	This message does not have a Beta portion.

b — Descriptor block number of the controllee; if zero, the data string is sent to the next lower controllee in the controllee chain.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

Figure 4-20. SEND A MESSAGE TO CONTROLLEE Message Format

GET MESSAGE FROM CONTROLLER OR OPERATOR (f=0016)

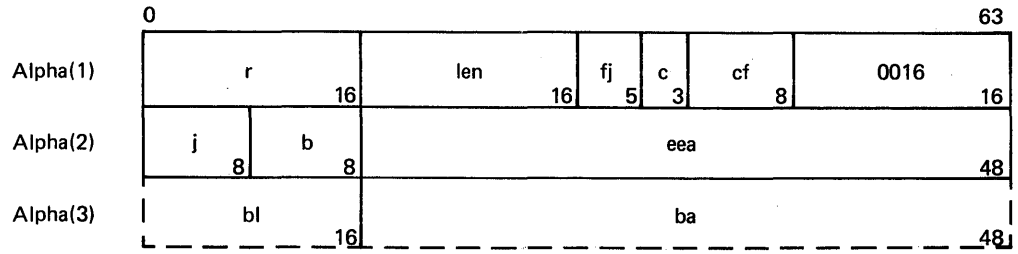
A string of binary or ASCII data sent by a controller program or the operator and waiting in a system buffer can be retrieved by this controllee program using a GET MESSAGE FROM CONTROLLER OR OPERATOR message. Depending on the message option selected, the data string being retrieved might be simply copied into Beta, or it could be processed into a set of symbols before it is stored into Beta. In any case, the data string being retrieved must not exceed 512 words (4096 character bytes).

Multiword symbols are permitted and processed without any special treatment. If the number of symbols exceeds the number requested, only the number requested are stored in Beta. If fewer symbols are returned than are requested, all symbols are stored in Beta. The operating system in this case never appends an end-of-message character.

Delimiters are always returned right-justified with null fill. Blanks are never treated as a special case (that is, if a space is a delimiter, all occurrences of blank result in a delimiter being returned; if space is not a delimiter, then spaces are processed the same as any other character).

A problem occurs if there is no data string waiting in the system buffer when a controllee (except for the immediate controllee of a level-1 task) issues this message. The controllee will stop running and wait for a data string from its controller. The next higher controller in the controllee chain will start running.

The format of this message is shown in figure 4-21. The Beta portion is discussed under the c field description.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, control proceeds normally and this field contains the number of words (if an unprocessed data string was obtained) or character bytes (if a data string processed into symbols was obtained) returned in Beta. If control returns to the error exit address, nothing was returned in Beta and the values are:

1	Count of bytes returned was either zero or greater than 4096
2	Illegal option was specified for this message
3	No controller or operator message existed
7	More than 200 delimiters are defined by this program
9	This message was issued from a level-1 task
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length in words (if the data string is to be translated into symbols) or character bytes (if an untranslated data string is to be obtained) and the virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length of the Beta portion in words or character bytes. Maximum is 512 words (4096 bytes).

fj — Fill and justification for the message. For all options except c=0, the values are:

0	Left justification, blank fill to the right
1	Left justification, zero fill to the right
4	Right justification, blank fill to the left
5	Right justification, zero fill to the left

c — Message options:

0	The data string is to be copied from the system buffer to Beta, beginning at Beta(1). If the number of words in the data string exceeds the number specified by the len field, only the first len words are copied to Beta. If there are fewer than the number requested, the last word of the data string is left-justified with binary zero fill.
1	The data string is to be translated into symbols. Delimiters must be defined by the program issuing this message, and their number must not exceed 200. Symbols are stored in Beta, one symbol per word, starting with Beta(2). Beta(1) contains the number of delimiters (leftmost 16 bits), and the virtual bit address of the delimiter buffer (rightmost 48 bits). Delimiters are stored left to right, character byte by character byte, in the buffer.
2	The data string is to be translated into symbols. Delimiters are blank, period, comma, slash, equals, plus, minus, and left and right parentheses. Symbols are stored in Beta starting with Beta(1).

Figure 4-21. GET MESSAGE FROM CONTROLLER OR OPERATOR Message Format (Sheet 1 of 2)

	3	The data string is to be translated into symbols. Delimiters are defined as installation parameter options. Symbols are stored in Beta starting with Beta(1).
cf	Control field. The values are:	
	0	If no data string from this program's controller is waiting in the system buffer, stop running this program until a data string arrives. Process and return the data string to Beta, and release the system buffer space occupied by the data string.
	1	If no data string from this program's controller is waiting in the system buffer, return control to the error exit address. Process and return the data string to Beta, and release the system buffer space occupied by the data string.
	2	If no data string from this program's controller is waiting in the system buffer, stop running this program until a data string arrives. Process and return the data string to Beta, but do not release the system buffer space occupied by the data string.
	3	If no data string from this program's controller is waiting in the system buffer, return control to the error exit address. Process and return the data string to Beta, but do not release the system buffer space occupied by the data string.
	4	If no data string from the operator is waiting in the system buffer, stop running the program until a data string arrives. Process and return the data string to Beta, and release the system buffer space occupied by the data string.
	5	If no data string from the operator is waiting in the system buffer, return control to the error exit address. Process and return the data string to Beta, and release the system buffer space occupied by the data string.
j	Level of the controller that sent the data string being retrieved; supplied by the operating system. If the data string came from the operator, this field is zero. If no data string was found in the system buffer, the operating system returns in j the level of the task that issued this message.	
b	Descriptor block number of the controller that sent the data string being retrieved; supplied by the operating system. (If the interactive processor was the controller, b is FF.) If the control field is 4 or 5, this field is the descriptor block number of the operator. If no data string was found in the system buffer, the operating system returns in b the descriptor block number of the task that issued this message.	
eea	Virtual bit address to receive control if an error occurs during processing of this message. If this field is zero when an error occurs, the error is considered fatal.	
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), this parameter indicates the length in words (if an untranslated data string is to be obtained) or character bytes (if the data string is to be translated into symbols) and virtual bit address of the first full word of the Beta portion.	

Figure 4-21. GET MESSAGE FROM CONTROLLER OR OPERATOR Message Format (Sheet 2 of 2)

GET MESSAGE FROM CONTROLLEE (f=0017)

A string of binary or ASCII data sent by a controllee program and waiting in a system buffer can be retrieved by this controller program using a GET MESSAGE FROM CONTROLLEE message. Depending on the message option selected, the data string being retrieved might be simply copied into Beta, or it could be processed into a set of symbols before it is stored in Beta. In any case, the data string being retrieved must not exceed 512 words (4096 character bytes).

Multiword symbols are permitted and processed without any special treatment. If the number of symbols exceeds the number requested, only the number requested are stored in Beta. If fewer symbols are returned than are requested, all symbols are stored in Beta. The operating system in this case never appends an end-of-message character.

Delimiters are always returned right-justified with null fill. Blanks are never treated as a special case (that is, if a space is a delimiter, all occurrences of blank result in a delimiter being returned; if space is not a delimiter, then spaces are processed the same as any other character).

The format of this message is shown in figure 4-22. The Beta portion is discussed under the c field description.

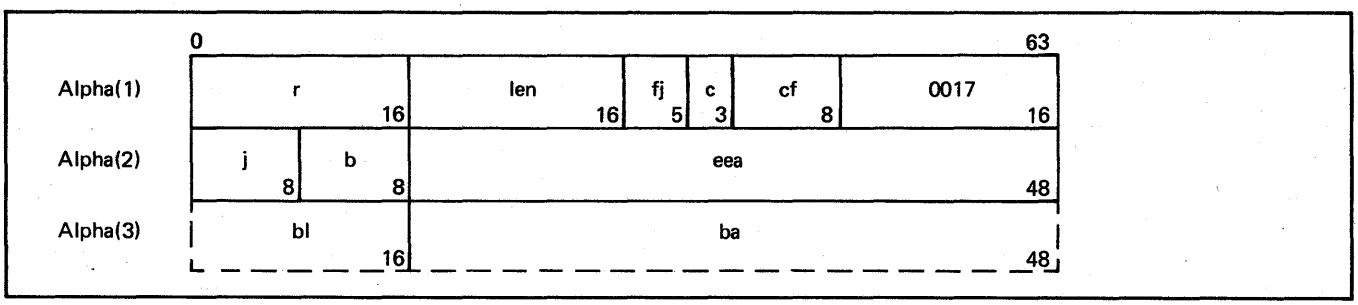


Figure 4-22. GET MESSAGE FROM CONTROLLEE Message Format (Sheet 1 of 2)

r — Response code; returned by the operating system when this message has been processed. If no error occurs, control proceeds normally and this field contains the number of character bytes (if a message was obtained) or words (if symbols were obtained) returned in Beta. If control returns to the error exit address, nothing is returned in Beta and the values are:

1	Count of bytes returned was zero or greater than 4096
2	Illegal option was specified for this message
3	No controllee message existed
4	Message from the controller (not the controllee) was waiting
6	This program started because the controllee, whose level and descriptor number are stored in fields j and b, is waiting for a message from the controller
7	More than 200 delimiters are defined by this program
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length in character bytes (if an untranslated data string is to be obtained) or words (if the data string is to be translated into symbols) and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length of the Beta portion in words or character bytes. Maximum is 512 words (4096 bytes).

fj — Fill and justification for the message. For all options except c=0, the values are:

0	Left justification, blank fill to the right	4	Right justification, blank fill to the left
1	Left justification, zero fill to the right	5	Right justification, zero fill to the left

c — Message format options:

0	The data string is to be copied from the system buffer to Beta, beginning at Beta(1). If the number of words in the data string exceeds the number specified by the len field, only the first len words are copied to Beta. If there are fewer than the number requested, the last word of the data string is left-justified with binary zero fill.
1	The data string is to be translated into symbols. Delimiters must be defined by the program issuing this message, and their number must not exceed 200. Symbols are stored in Beta, one symbol per word, starting with Beta(2). Beta(1) contains the number of delimiters (leftmost 16 bits), and the virtual bit address of the delimiter buffer (rightmost 48 bits). Delimiters are stored left to right, character byte by character byte, in the buffer.
2	The data string is to be translated into symbols. Delimiters are blank, period, comma, slash, equals, plus, minus, and left and right parentheses. Symbols are stored in Beta starting with Beta(1).
3	The data string is to be translated into symbols. Delimiters are defined as installation parameter options. Symbols are stored in Beta, starting with Beta(1).

cf — Control field. The values are:

0	After data string has been retrieved, release system buffer space occupied by string.
2	After data string has been retrieved, do not release system buffer space occupied by string.

j — Level of the controllee that sent the data string; supplied by the operating system. The value in this field has no meaning if the controllee that sent the data string being retrieved has been disconnected.

b — Descriptor number of the controllee that sent the data string; supplied by the operating system. The value in this field has no meaning if the controllee that sent the data string being retrieved has been disconnected.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in words (if an untranslated data string is to be obtained) or character bytes (if the data string is to be translated into symbols) and virtual bit address of the first full word of the Beta portion.

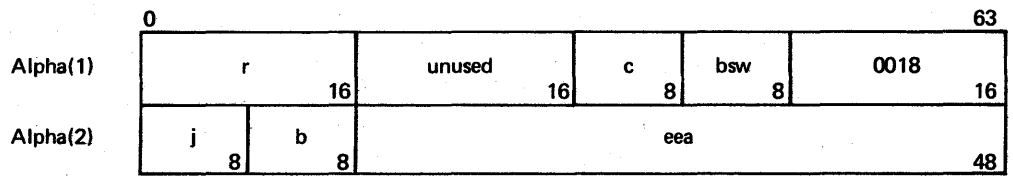
Figure 4-22. GET MESSAGE FROM CONTROLLEE Message Format (Sheet 2 of 2)

MESSAGE CONTROL (f=0018)

A user program can issue this message to inform the operating system that data strings directed to the program should be sent to another controllee or controller in the chain. A controllee can direct data strings to a terminal by setting the b field to FF. Data strings sent to the user program specifically by descriptor block number cannot be redirected with a MESSAGE CONTROL message.

If the input bypass is set, controller data strings not specifically directed to the user program are sent to the program's controllee. If the output bypass is set, controllee data strings not specifically directed to the user program are sent to the program's controller.

The format of this message is shown in figure 4-23. The message consists of an Alpha portion only.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero. If the response code is 1, no controller or controllee has that descriptor block number.

c — Options for bypassing input and output messages:

1	Manipulate the input bypass according to control switch bsw
2	Manipulate the output bypass according to control switch bsw
3	Manipulate both the input and output bypasses according to control switch bsw
4	Determine the message destination from the controllee descriptor block number in field j
5	Determine the message destination from the controller descriptor block number in field b
6	Examine both field b and field j to determine the message destination

bsw — Bypass control switch. If this field is zero, turn off the bypass switch; if this field is 1, turn on the bypass switch.

j — Descriptor block number of the destination controllee.

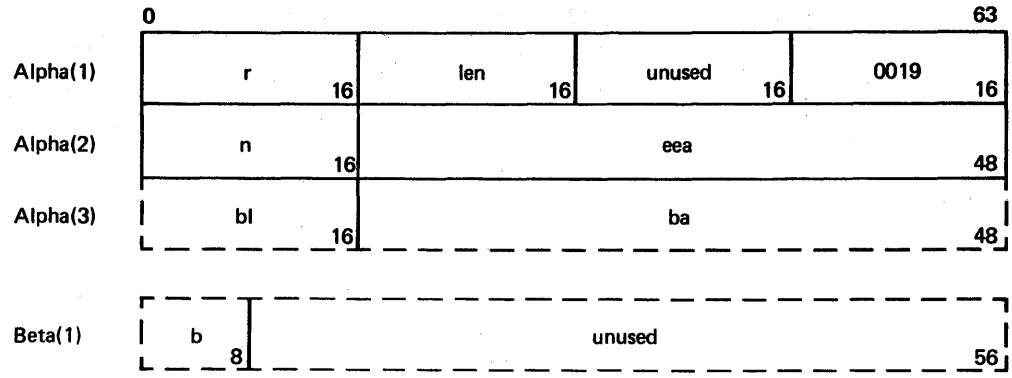
b — Descriptor block number of the destination controller.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

Figure 4-23. MESSAGE CONTROL Message Format

REMOVE CONTROLLEE FROM MAIN MEMORY (f=0019)

A controller (the user program) can swap a controllee program or itself from main memory to mass storage. The controller program stops running until all controllee pages are written to mass storage. The format of this message is shown in figure 4-24. For option zero, the program issuing this message must have only one controllee.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	No controllee exists with this descriptor block number
3	More than one controllee for an n field value of zero
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the len field is zero (n=1 only)

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. This field is 0 or 1.

n — Specifies the controllee to be removed. The values are:

0	Remove the controllee program to mass storage; no Beta portion is required for this option
1	Remove the controllee specified in Beta(1) to mass storage
2	Remove this program to mass storage; no Beta portion is required for this option

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

b — Descriptor block number of the controllee to be removed to mass storage.

Figure 4-24. REMOVE CONTROLLEE FROM MAIN MEMORY Message Format

SEND A MESSAGE TO OPERATOR (f=001A)

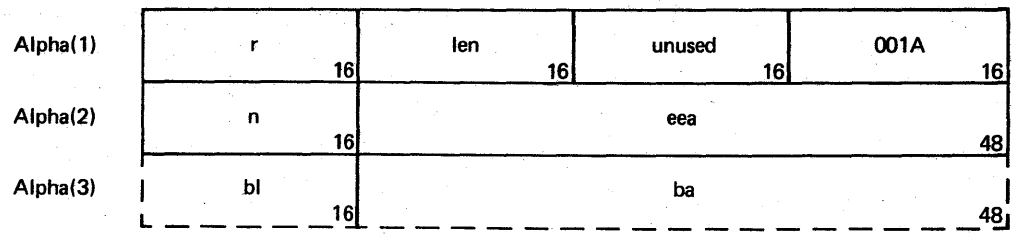
A program uses this message to send a string of binary or ASCII data to the operator. The system copies the data string from the Beta portion of the system message to a system buffer.

If the system buffer is full or the operator is not logged in, the string cannot be sent. If the buffer is full, the system continues task execution at the error exit address. If the operator is not logged in, the system either suspends task execution until the operator logs in (n=0 or 2) or continues task execution at the error exit address (n=1 or 3).

Because the operator of a busy system could miss a string sent to him, the system provides a string save table. If the operator is logged in and n=2 or 3, the string is kept in the string save table. Only one string per task is kept. If the task sends another string, only the most recent string requiring a response is kept.

The operator can access the string save table to see the most recent strings sent by executing tasks. The operator clears a string from the save table with the command TELL DB. (Strings are kept by descriptor block number.)

The format of this message is shown in figure 4-25. The Beta portion contains the string being sent; maximum length of the string is 80 character bytes (10 words).



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

- 1 — Specified length of the Beta portion in character bytes was out-of-range (either zero or greater than 80)
- 2 — String in Beta could not be sent to the operator because no system buffer was available or the operator was not logged on; error processing is indicated by the value of the n field
- 214 — Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the len field is zero

len — If this field is #FFFF, Alpha(3) contains the length in character bytes and the virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len if the length in character bytes of the Beta portion. The len field should be greater than zero and less than or equal to 80.

n — Indicates action to be taken if the string cannot be sent; also indicates whether the string should be kept in the save table.

- 0 — If the operator is not logged in, suspend task execution until the operator logs in. If the operator is logged in but the system buffer is full, continue execution at the error exit address.
  Do not enter the string in the save table.
- 1 — If the operator is not logged in or the system buffer is full, continue execution at the error exit address.
  Do not enter the string in the save table.
- 2 — If the operator is not logged in, suspend task execution until the operator logs in. If the operator is logged in but the system buffer is full, continue execution at the error exit address.
  Enter the string in the save table.
- 3 — If the operator is not logged in or the system buffer is full, continue execution at the error exit address.
  Enter the string in the save table.

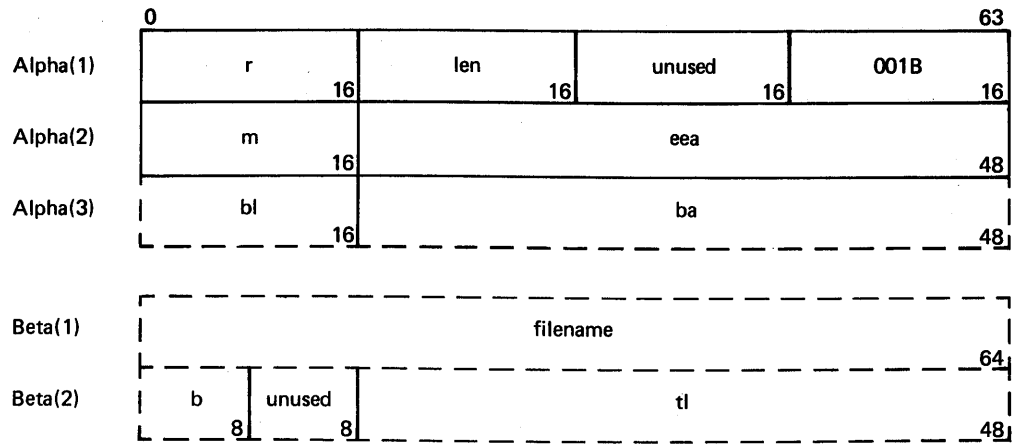
eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-25. SEND A MESSAGE TO OPERATOR Message Format

INITIALIZE OR DISCONNECT CONTROLLEE (f=001B)

A user program can make another program a controllee, and optionally start the controllee running, by using the INITIALIZE OR DISCONNECT CONTROLLEE message. It also can be used to disconnect a previously connected controllee. Up to eight levels of program controllees are permitted in a controllee chain, making a possible total of up to nine levels. The format of this message is shown in figure 4-26.



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Controllee program is already present
2	Illegal message option
3	Controllee program file was not found or not attached
4	Insufficient time to run the controllee program
5	Illegal priority
6	Error in creating the drop file
7	Controllee program file is not executable
8	Mass storage device error
9	Full system tables inhibit initialization of the controllee program at this time
A	Abnormality in the controllee program file or drop file input/output connector entry
B	Nine task levels are present already
C	No controllee is present (for disconnect only); if a controllee program is being disconnected, this response code indicates no controllee program is present
D	Controllee program drop file is too small
E	Unable to destroy the existing drop file
F	Unable to restart the controllee because the interrupt register table is full
10	Drop file cannot be verified
11	C500 request error
12	Bad minus page in the controllee file
13	Undefined error in the drop file verification
14	Controllee program file is privileged open
15	No FST space.
16	No drop file pack is logically up.
17	IOC for file not found.
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. The length of the Beta portion is at least 2.

m — Message options:

0	Initialize the controllee program and restart this program
1	Initialize the controllee program and immediately begin running it; stop running this program
10	Disconnect the controllee program and restart this program (the Beta portion of the message is required for this option, but is not used)

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-26. INITIALIZE OR DISCONNECT CONTROLLEE Message Format (Sheet 1 of 2)

filename	Name of the controllee program (must be a virtual code file), left-justified with blank fill.
b	Controllee program's descriptor block number; returned by the operating system when the message is issued. If the controllee program is disconnected and reconnected, this number could change.
tl	Time limit for the controllee program, in microseconds. When this field is zero, the controller's time limit is used.

Figure 4-26. INITIALIZE OR DISCONNECT CONTROLLEE Message Format (Sheet 2 of 2)

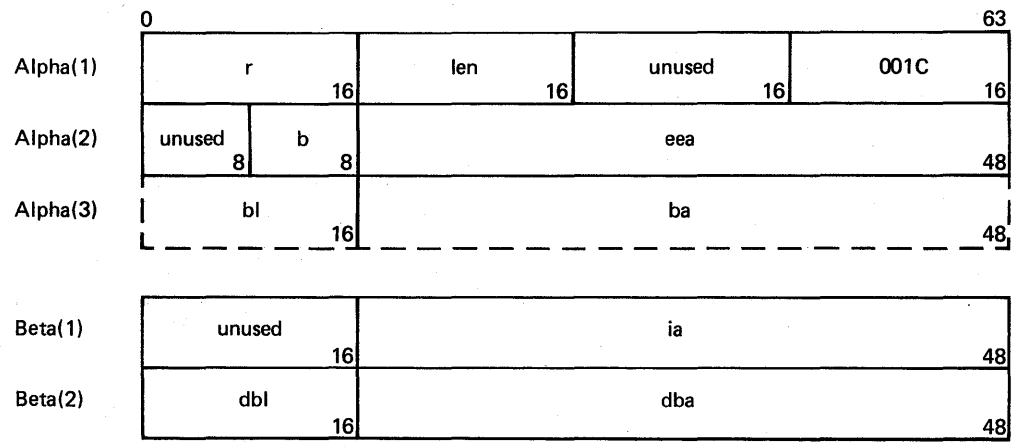
PROGRAM INTERRUPT (f=001C)

The operating system supports one level of software interrupt for any program. With the PROGRAM INTERRUPT message, a user program can tell the operating system whether or not the program can be interrupted. If interrupts are enabled, an ASCII character string must be waiting at the interrupt address specified in Beta(1) of the PROGRAM INTERRUPT message at the time control passes to the interrupt address. For control to return to the calling routine, the interrupt routine must issue a RETURN FROM INTERRUPT message when it has finished performing its tasks.

When a program is interrupted, the program's minus page is altered before control is passed to the virtual address specified by the user in a PROGRAM INTERRUPT message. The minus page has space for the current invisible package (level 0). The interrupt register table has space for the interrupted routine invisible package (level 1). (These level designations are not to be confused with the level of a task in the controllee chain, nor with the security level of a task.) At the time of an interrupt, the level-1 invisible package becomes the current execution invisible package (level 0), and the level-0 invisible package is saved in the interrupt register table. The operating system saves the register file image for the old level 0, and places in register 3 a pointer to the Alpha portion of, and an index to the Beta portion of, the message that caused the interrupt (an EXPLICIT I/O message, for instance). The operating system also puts into register 1E the length and address of the data base to be used by the interrupt routine. Initializing the rest of the register file is the responsibility of the interrupt routine.

When message option 1 is specified, any ASCII character string preceded by (sc)I that is received from a terminal interrupts the user program (the currently executing program). The symbol (sc) is a special character defined by the installation. (See volume 1 of this manual.) When the string has been received, the (sc)I preceding the string is stripped and the string is realigned at the beginning of the word. An (sc)I interrupt causes any outstanding output message to be released to the output device. When (sc)I precedes a string, the message interrupts the highest level controller that issued a PROGRAM INTERRUPT message with message option 1 (highest level refers to level in the controllee chain).

The format of the PROGRAM INTERRUPT message is shown in figure 4-27.



r	Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:
	1 — Value of the interrupt address is greater than the upper limit of the virtual bit address range
	2 — Program selected an illegal interrupt option
	214 — Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified
len	If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion.

Figure 4-27. PROGRAM INTERRUPT Message Format (Sheet 1 of 2)

b	Interrupt options: 0 This program can be interrupted by any program 1 This program can be interrupted by a terminal if the data at the interrupt address begins with the two characters (sc)I 2 This program must not be interrupted When this message is issued for options 0 or 1, the program issuing this message can be interrupted by all subsequent messages and interrupts coming from a terminal until this program either issues this message with option 2 or terminates.
eea	Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.
ia	Interrupt address, the virtual bit address of the word to which control transfers upon occurrence of an interrupt.
dbl	Length of the data base to be established if an interrupt occurs.
dba	Address of the data base to be established if an interrupt occurs. If this field is zero, the data base of the interrupted program (the program issuing this message) is used.

Figure 4-27. PROGRAM INTERRUPT Message Format (Sheet 2 of 2)

INITIALIZE CONTROLLEE CHAIN (f=001D)

A user program can issue this message to make itself the controller of a chain of controllees. Up to nine levels of controllee programs are permitted in any controllee chain (for example, if the program issuing this message is the controllee of a level 1 task, a maximum of seven controllees can be specified in this message). Control is always returned to the user program after the call has been processed; unlike the INITIALIZE OR DISCONNECT CONTROLLEE message (function 1B), this message cannot be used to start a controllee running.

Any error in the request causes the entire chain to be ignored, and none of the controllees are initialized. The format of this message is shown in figure 4-28.

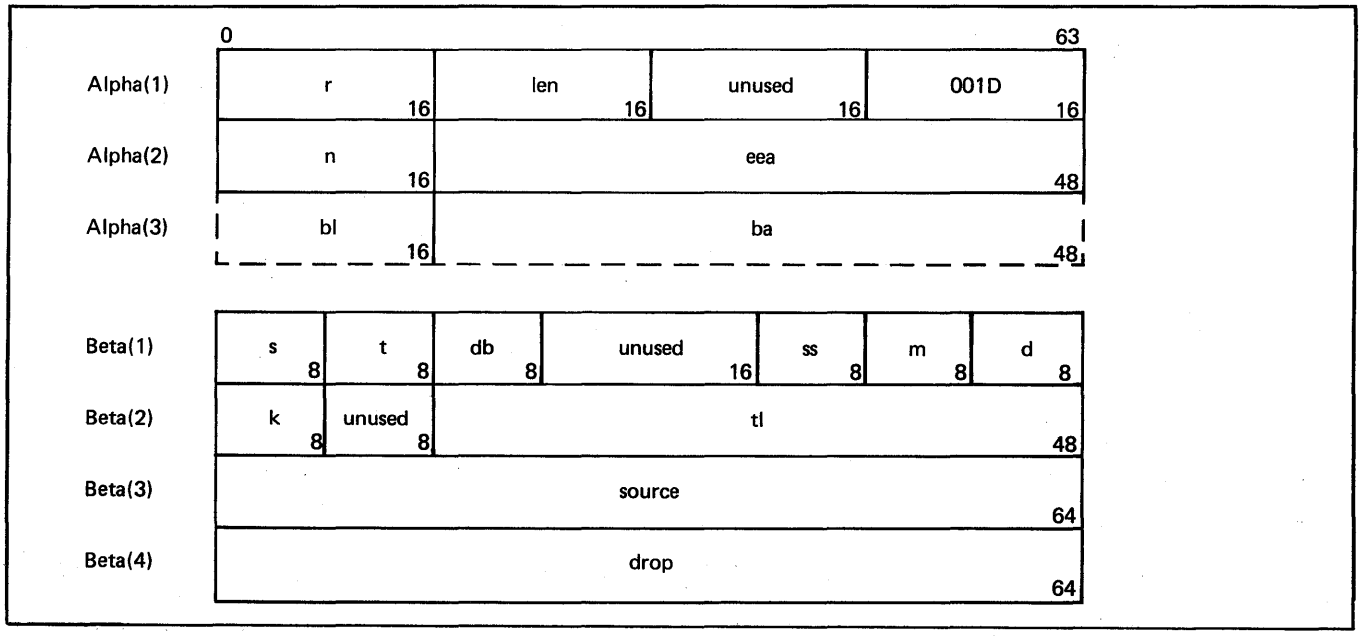


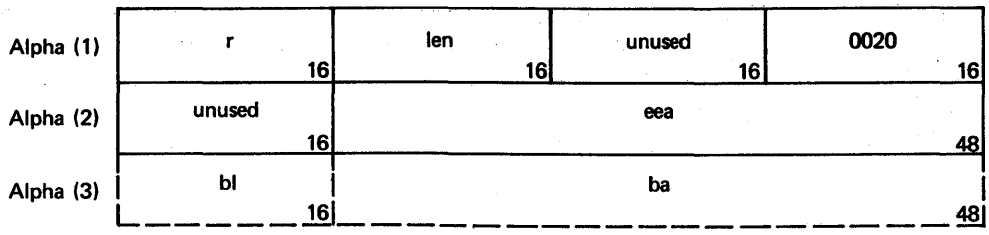
Figure 4-28. INITIALIZE CONTROLLEE CHAIN Message Format (Sheet 1 of 2)

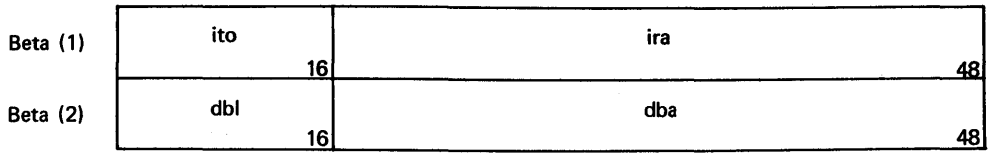
r	Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise: 1 Controllee program is already present 2 Full system tables inhibit initialization of the controllee program at this time 3 Controllee chain levels exceed nine 4 Error in an attempt to initialize the controllee; see an ss field in Beta 5 Controllee file is privileged-open
len	If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion (a multiple of 4, and always less than or equal to 14).
n	Number of tasks to be initialized in the chain; n is always less than or equal to 4.
eea	Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.
s	Controllee chain level (2 through 9) of the program named in Beta(3); returned by the operating system. Beta words appear in the order in which the controllees are to be initialized, highest to lowest in the chain.
t	Descriptor block number of the program named in Beta(3); returned by the operating system.
db	Descriptor block number of the controller of the program named in Beta(3); returned by the operating system.
ss	File initialization error. The values are: 3 Controllee program file was not found or was not attached 4 Insufficient time to run the controllee program 6 Error in creating the drop file 7 Controllee program file is not executable 8 Mass storage error 9 Abnormality in the controllee program file or drop file input/output connector entry B No FST space C No drop file pack was logically up D IOC for file not found F Controllee program file is privileged-open
m	Task level, relative to the controllee specified in Beta(3), to receive messages from higher in the chain.
d	Task level, relative to the controllee specified in Beta(3), to receive messages from lower in the chain.
k	Descriptor block number of the controllee (in this chain) of the program named in Beta(3). This field is zero if there is no controllee.
tl	Amount that remains in microseconds, of the time allowed for running the controllee program. When the controllee has exhausted the time, this field is zero.
source	ASCII name of the executable source file to be initialized if the drop field is zero, left-justified with blank fill. Returned by the operating system if the drop field is not zero.
drop	ASCII name of the drop file, left-justified with blank fill. If the user provides a nonzero value in this field, the program is started from this drop file (the drop file contains an input/output connector containing the source file name associated with the drop file).

Figure 4-28. INITIALIZE CONTROLLEE CHAIN Message Format (Sheet 2 of 2)

ENABLE/DISABLE ABNORMAL TERMINATION CONTROL (ATC) (f=0020)

The ENABLE/DISABLE ATC message allows the user to process what is normally a fatal error. The user does this by setting/zeroing the interrupt subroutine address and data base descriptor in the minus page. If ATC is already enabled, the user may reissue this message to change the interrupt routine information. However, all of the fields must be supplied as if this message had not previously been issued. The format of the message is shown in figure 4-29.





r — Response code returned by operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

61	Entry point or data base address out of virtual address range for user program or hexadecimal bit address less than 8000
62	Interrupt type option is illegal
63	Data base length out of range
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If len=#FFFF, Alpha(3) contains the length (bl) and virtual bit address (ba) of the Beta portion. Otherwise, Beta is assumed to begin at Alpha(3) and len is the length of the Beta portion (two words).

eea — Virtual bit error exit address to receive control if an error occurs during message processing (r≠0). If eea=0, the error is fatal.

bl — Beta length, if len=#FFFF.

ba — Virtual bit address of the Beta portion's first full word, if len=#FFFF.

ito — Interrupt type option:

0	Enable ATC
1	Disable ATC

ira — User interrupt subroutine address (virtual bit address) to be entered if predefined system errors occur.

dbl — Data base length (in words) of user's interrupt subroutine.

dba — Data base bit address of user's interrupt subroutine.

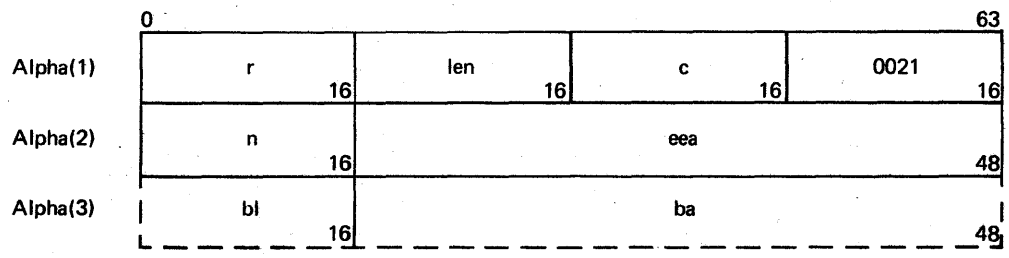
Figure 4-29. ENABLE/DISABLE ABNORMAL TERMINATION CONTROL Message Format

EXECUTE OPERATOR COMMAND (f=0021)

This message can be issued only by a privileged user and executes exactly one of the operator commands that are listed as possible values for the c field in Alpha(1). The user number of the issuer must be the operator's user number, 999998.

The format of the Alpha portion of this message is shown in figure 4-30. The Beta word formats depend on the message option (c field) in Alpha(1), and are shown in figure 4-31. Each option is a correspondence between the Beta format and the operator command. (See the CYBER 200 Operator's Guide).

A descriptor block number uniquely identifies a program in the system. The number is an index into the system's descriptor block (DB) table. (See internal documentation for a description of the DB table.)



r — Response code; returned by the operating system when this message has been processed.

0	No errors
1	Illegal option (c field value is out-of-bounds, or there is a bad parameter)
2	User is not privileged and is not running under the operator's user number
3	Error in a resident system call issued by the virtual system routine OPCOM
4	Same as 3

len — This field must always be #FFFF, indicating that Alpha(3) contains the length and virtual bit address of the Beta portion of the message.

c — Message options: Each of the options has a different Beta format. The Beta formats are described following the Alpha field definitions.

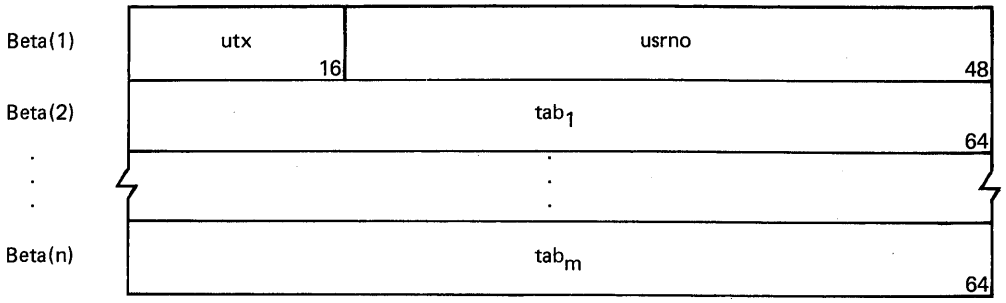
1	Display user
2	Display files
3	Display system
4	Display one task
5	Tell (send a message to the users)
6	Break (terminate the task)
7	Drop (terminate the task and logoff the user)
8	Set a parameter
9	Assign a unit
A	Display memory
B	Set memory (enter memory)
C	Turn the unit on or off
D	Date
E	Time
F	Status (obtain the status of the disk or tape)
10	No op (reserved for terminate operator)
11	Display all tasks
12	Display the timepool (repository)
13	Delete the user or division
14	Create a new user
15	Modify the user directory
16	Suspend or resume the task
17	Modify the account (delete, replace, create, or list the account)
18	Get the operator's teletypewriter terminal unit (TTY) number
19	Turn output processing on or off
1A	Abort the tape request
1B	Display up to five messages to the operator
1C	Checkpoint files
1D	Turn on or off the no logon flag
1E	Put comment in dayfile
1F	Set job category priority.
20	Evict job from input queue.
21	Submit job to CPU scheduler.
22	Free job for submission to CPU scheduler
23	Hold job in input queue, preventing its submission to the CPU scheduler
24	Turn on or turn off job submission to the CPU scheduler
25	Set maximum large page limit for job category
26	Set maximum memory overcommitment percentage
27	Set maximum combined time limit for all executing jobs
28	Set maximum job time limit for job category
29	Set maximum working set size limit for job category
2A	Set maximum executing jobs for job category
2B	Create or delete job category validation for user
2C	Retrieve H display information
2D	Retrieve V display information
2E	Change priority or time limit for job in input queue
2F	Retrieve validated job categories for user number

Figure 4-30. EXECUTE OPERATOR COMMAND Message (Alpha Format) (Sheet 1 of 2)

n	The number of words in the Beta portion of the message, the value of which depends on the message option (c field). (Refer to the specific c option description.)
eea	Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.
bl, ba	If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-30. EXECUTE OPERATOR COMMAND Message (Alpha Format) (Sheet 2 of 2)

c=1 Obtain user table information about one user or all users who are currently on the system. The value of the n field in Alpha(2) is returned by the operating system. The value of n is #FFFF if m is 5, is 18 if m is 16, and is m otherwise.

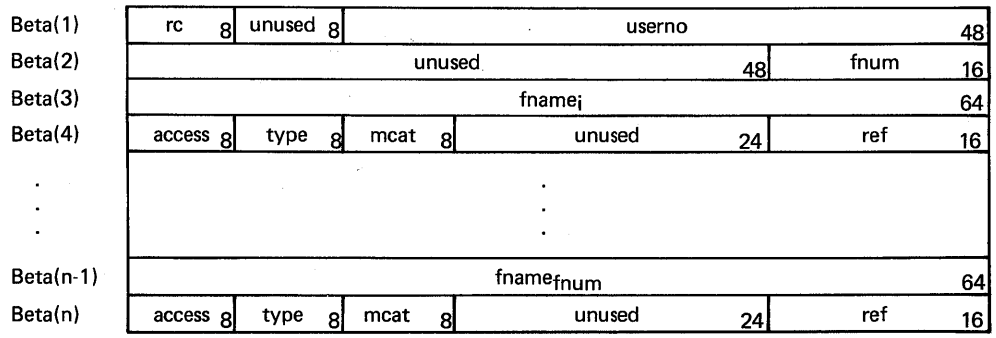


utx Type of information in the tab field; returned by the operating system. If this field is zero, the information is from the user directory. If this field is nonzero, it is from the user table; the utx field contains the user table index of the table entry.

usrno Six-character ASCII user number of the user table or user directory entry in the tab fields; or, the character string ALL, left-justified with blank fill, indicating that user numbers and TTY numbers of all logged-on users are to be placed in the tab field. Supplied by the user.

tab_i The tab_1 field contains binary zeros if no user identified by the usrno field was found. Otherwise, tab_1 contains the first word of a 16-word copy of the user table entry (utx≠0) for an active user; a 5-word copy of the first part of the user directory entry (utx=0) for an inactive user; or m words, one for each currently logged-on (active) user, each of which contains the binary user number in the leftmost 48 bits and a TTY number in the rightmost 16 bits.

c=2 Return a list of the private files belonging to the specified user. This option also returns the attributes of each file as stored in the FILEI table.



rc Response code returned by the system.

0	User is active.
≠ 0	User is inactive (his files are not in FILEI).

usrno User number (6 ASCII characters).

fnum Number of file names returned by the system.

$fname_i$ File name (1 to 8 ASCII characters, left-justified, blank-filled).

acces Default file access permission.

0 or 4	No access allowed.
1 or 5	Write access allowed.
2 or 6	Read access allowed.
3 or 7	Read or write access allowed.

type File type.

0	Physical file.
1	Virtual data file.
2	Virtual code file.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 1 of 16)

mcat File management category.

0	Mass storage file.
1	Scratch file.
2	Output file.
3	Write-temporary file.
4	Magnetic tape file.
5	User-generated drop file.
6	System-generated drop file.
7	Batch job file.

ref Number of times this file has been opened.

c=3 Return the statistics table, number of users logged on, microsecond clock time, and number of entries in the descriptor block table.

Beta(1) Microsecond central processor clock (zero at power on).

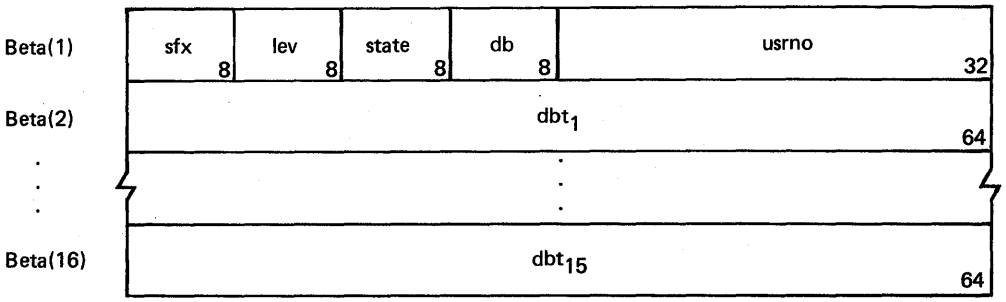
Beta(2) Number of users logged on.

Beta(3) Number of descriptor block entries.

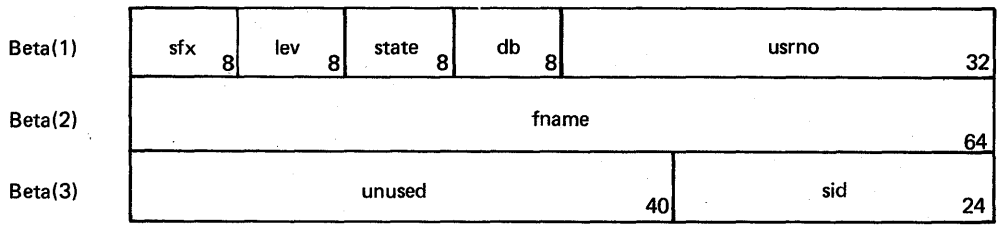
Beta(4) to Beta(82) Copy of the statistics table.

c=4 Return information about a specified active task, or for all active tasks. The operating system returns 16 Beta words when one active task is to be displayed (sfx=0); otherwise, three Beta words are displayed for each active task.

Beta portion when the sfx field is not zero:



Beta portion format when the sfx field is not zero:



sfx If all tasks are to be displayed, this field must be zero; otherwise, one task is to be displayed, and the suffix under which it is running must be given in this field:

1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

lev Controllee level number.

state Program state of the task (see appendix F).

db Descriptor block number of the task whose descriptor block table entry follows immediately in the next 15 words.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 2 of 16)

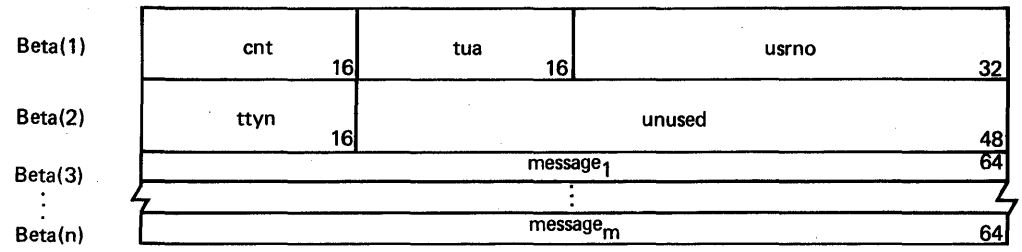
usrno — Binary user number under which the task is executing.

fname — ASCII source file name associated with the db field, left-justified with blank fill. fname equals 0 if file is not found.

sid — Three-character mainframe identifier; set only if the task is in a waiting-for-mainframe state (state=14).

dbt_i — A word of a 15-word descriptor block table entry returned by the operating system when one task is to be displayed.

c=5 — Transmit a specified character string to the output buffer of one or all users; or, respond to retained messages or to a message that indicated an invalid tape label during tape assignment.



cnt — Number of characters in the message.

tua — Destination of the message. The values are:

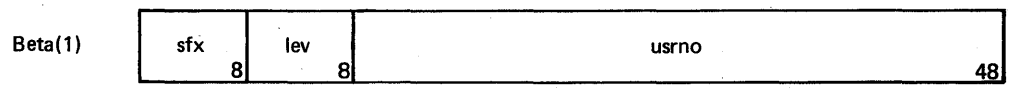
0	All users on the system
1	TTY specified in the usrno field
2	User specified in the usrno field
3	DB number (when responding to a message) specified in the usrno field

usrno — Binary TTY number (for tua=1), or binary user number (for tua=2), or binary descriptor block number (for tua=3).

ttyn — User's TTY number (for tua=2), returned by the operating system; or zero if the user or users are not logged on. Not used if the tua field is zero, 1, or 3.

message — A word of a TELL DB formatted message (see the CYBER 200 Operator's Guide). The message is in ASCII.

c=6 — Terminate the task and all its controllees, but do not log off the user (interactive) and do not terminate the entire job (batch).



sfx — Suffix under which the task is running (set by the user):

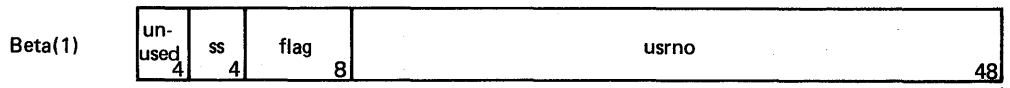
1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

The operating system zeros the rightmost four bits of this field if the task is found, or leaves the suffix in the field if the task is not found.

lev — Level of this task in the controllee chain; set by the user.

usrno — Binary user number for the task to be terminated.

c=7 — Terminate all tasks, all interactive tasks, or terminate one task and all its controllees, and log off the user (interactive) or terminate the job (batch).



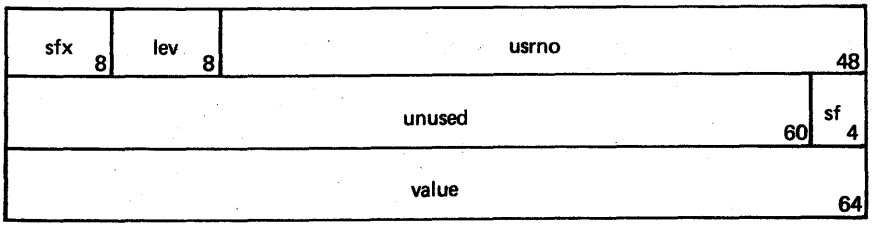
ss — Error response field. The operating system sets this field to zero if the task is found, or to nonzero if the task is not found.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 3 of 16)

flag — If flag 31 terminate the user, but rerun the batch job.

usrno — Binary user number for the task to be terminated, zero to indicate all tasks, or #FFFFFFFFFFFF to indicate all interactive tasks.

c=8 — Set the time limit, security level, or priority. (SET operator command).



sfx — Suffix under which the task is running:

1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

The operating system returns a return code in the rightmost 4 bits of this field (the suffix is overwritten). The values are:

0	Task is found
1	Task is not found
2	Value is out-of-range
3	Subfunction is out-of-range
4	Another request was changing the time limit when this request was processed.

lev — Level number for this task in the controllee chain.

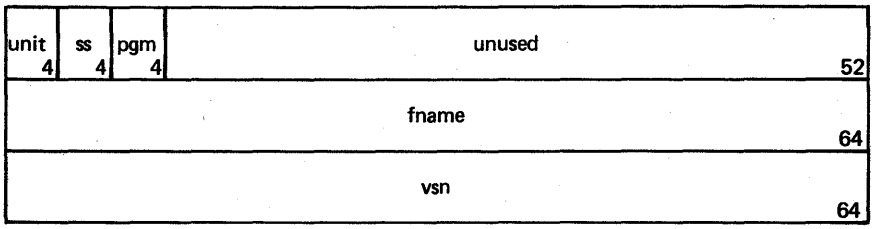
usrno — Binary user number for this task.

sf — Subfunction, specifying the parameter whose value is being set with this message, as follows:

2	Security level
3	Time limit
4	Priority

value — Value to which the parameter is to be set, in binary.

c=9 — Assign a magnetic tape unit to the specified logical device.



unit — Logical unit number to be assigned.

ss — Error response field. The operating system sets this field to one of the following values:

0	Unit is successfully assigned
1	Unit is already assigned
2	Wrong program
3	Unit is not available to the system

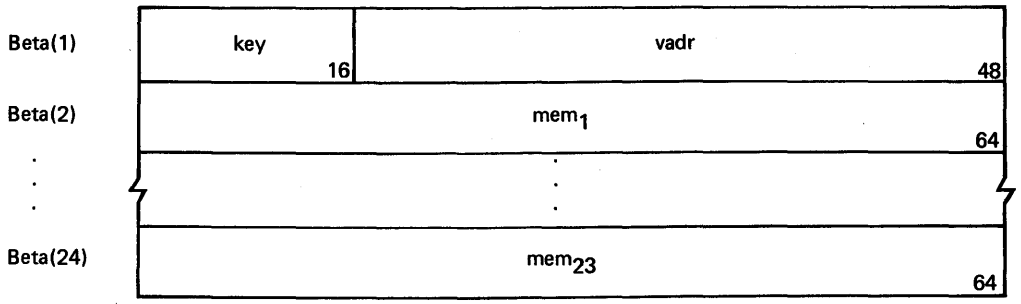
pgm — Descriptor block number of the task.

fname — Logical device name (tape file name) in the format described in section 3.

vsn — Volume serial number of the tape.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 4 of 16)

c=A Display 23 words of virtual system memory. The n field in Alpha(2) contains the number of words of virtual system memory plus 1 (n=24).

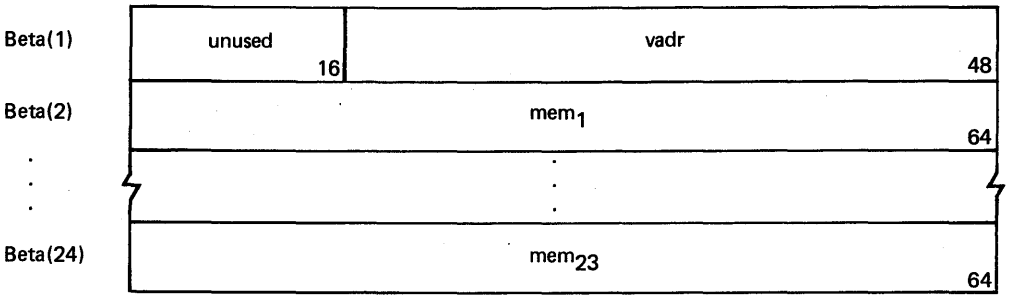


key Currently set to 1.

vadr Virtual bit address of the first word of the memory portion to be displayed.

mem_i A word of data retrieved from memory.

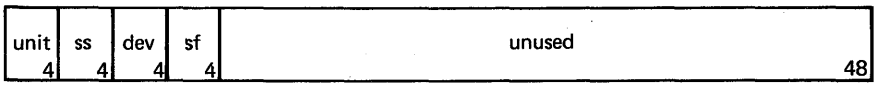
c=B Enter words into 23 words of virtual system memory. The n field in Alpha(2) must be set by the user to equal the number of words to be entered, plus 1 (n=24).



vadr Virtual bit address of the first word of the memory portion to be defined.

mem_i A word of data to be stored in memory.

c=C Turn the disk or tape unit off or on (logically, not physically).



unit Binary unit number; can be obtained by issuing a c=12 option first.

ss Error response field. The operating sets this field to one of the following values:

0	Function is completed successfully
1	Unit does not exist
2	Unit is assigned
3	Unit is unavailable
4	Unit is already in the requested state

dev Device type. The user sets this field to zero for a tape unit, or to 1 for a disk unit.

sf Subfunction. The user sets this field to zero to turn on the unit, or to 1 to turn off the unit.

c=D Set or request the date.

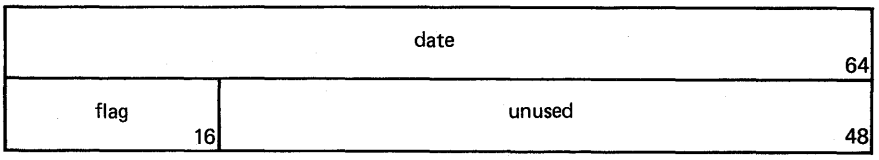
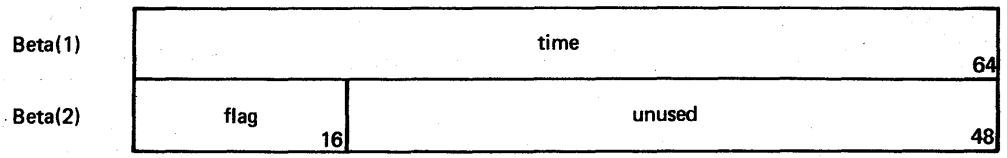


Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 5 of 16)

date — ASCII date, in the format mm/dd/yy (month/day/year); set by the user if the flag field is 1, or set by the operating system if the flag field is zero.

flag — This field must be zero for the data to be returned, or 1 to set the date.

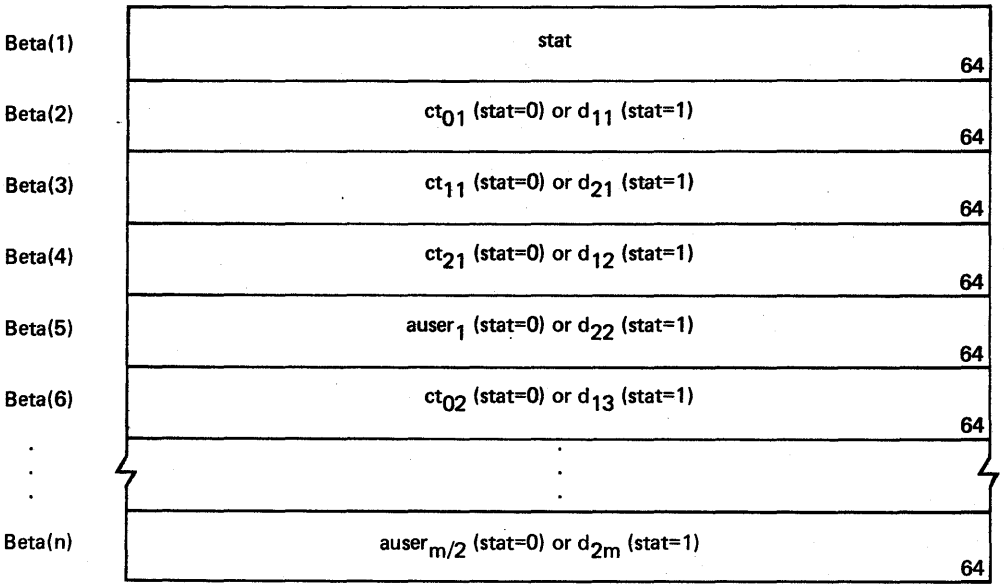
c=E — Set or request the time.



time — ASCII clock time, in the format hh.mm.ss (hour.minutes.seconds); set by the user if the flag field is 1, or set by the operating system if the flag field is zero.

flag — This field must be zero for the time to be returned, or 1 for the time to be set.

c=F — Return the disk or tape status information. The operating system returns the number of disks times 2 or tapes times 4 in the n field of Alpha(2).



stat — Function to be performed. The user must set this field to zero to get the tape status for all tape units on the system, or set it to 1 to get the disk status for all disk units on the system. The operating system returns the number of four-word entries (for tapes) or two-word entries (for disks) in the stat field.

ct_{ij} — The i^{th} word of the CORTAB table entry (when the stat field is zero) for the j^{th} tape unit. (See internal documentation for a description of the CORTAB table.)

$auser_i$ — Eight-character ASCII user number of the device user (if any), right-justified with blank fill.

d_{ij} — The i^{th} word of the DISF table entry (when the stat field is 1) for the j^{th} disk unit. (See internal documentation for a description of the DISF table.)

c=11 — Return information about all active tasks in the system. This is the all-tasks counterpart of the EXECUTE OPERATOR COMMAND message option 4. Two Beta words are returned for each task; the operating system sets the n field in Alpha(2) to the number of tasks in the system.

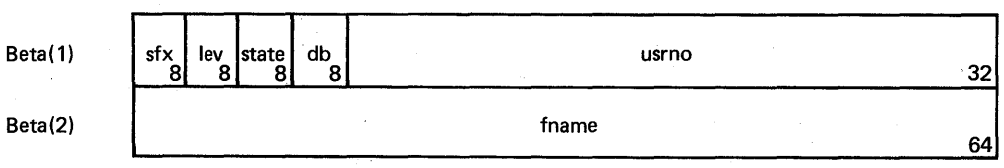


Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 6 of 16)

sfx — Suffix under which the task is running:

1	Suffix A
2	Suffix B
3	Suffix C
4	Suffix D

lev — Level of the task in the controllee chain.

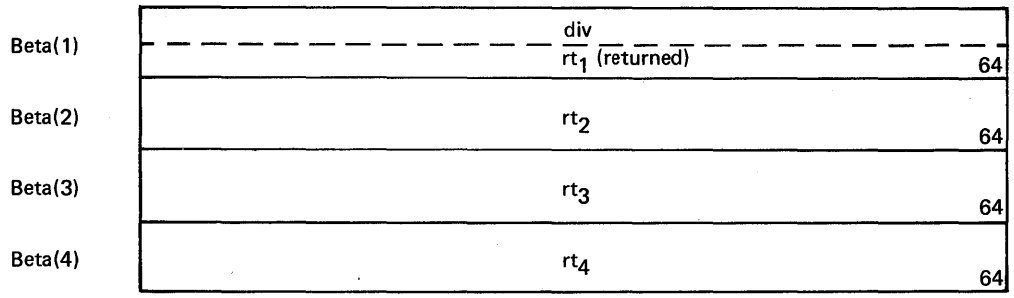
state — Program state of the task (see appendix F).

db — Descriptor block number for the task.

usrno — Binary user number under which the task is running.

fname — Source file name of the task, in ASCII.

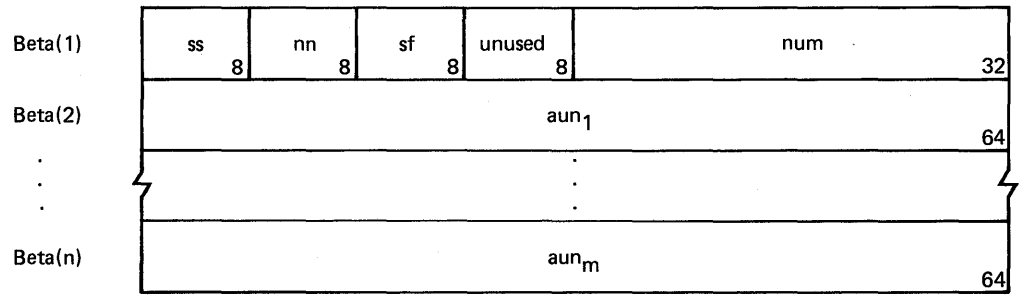
c=12 — Display information about the repository of a specified division.



div — Division number, in binary, of the repository to be displayed. The operating system sets this field to all zeros if the division specified by the user cannot be found.

rt_i — The i^{th} word of the repository table entry for the indicated division; returned by the operating system. (The table entry format is given in section 6.)

c=13 — Delete the user, or division of users, from the operating system. The users being deleted must not be logged in at the time the message is issued; if they are, the function is not performed.



ss — Error response field. The values are:

0	User or division was successfully deleted
1	User is active, was not deleted
2	User was not found
3	Division was not found
4	Users are active in the division
5	Illegal subfunction value

nn — Number of users active; returned only when the error response field is 4.

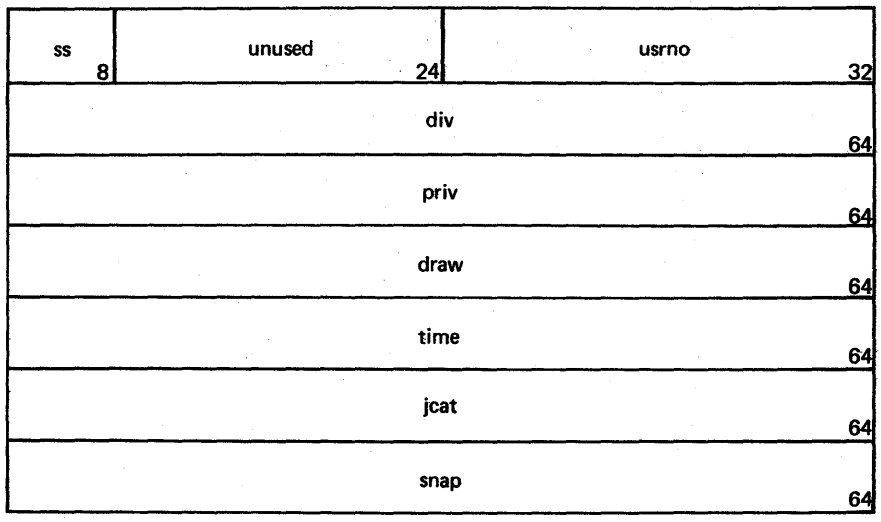
sf — Subfunction. This field is set to 1 to delete a user, or set to 2 to delete a division.

num — Binary user number of ASCII division number of the user or division to be deleted. The division number must be right-justified with blank fill.

aun_i — ASCII user number of an active user, right-justified with blank fill; returned only when the error response field is 4.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 7 of 16)

c=14 Create a new user number and specify the division to which the user belongs, privileged status, priority status, percentage draw on the respository allowed, initial private money, and whether or not unused money should be returned to the private account. All fields except the error response field must be supplied.



ss Error response field.

0	User successfully created.
1	User already exists.
2	Job category specified does not exist.

usrno Binary user number of the user to be created.

div ASCII division number for the new user, right-justified with blank fill.

priv This field must be set to zero to indicate the user is not privileged, or nonzero to indicate the user is privileged.

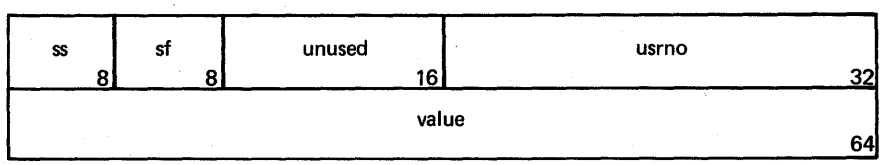
draw Percentage draw to be allowed from the repository, 0 to 100.

time Initial private quantity of STU units.

jcat Job category (8 ASCII characters, left-justified, blank-filled).

snap This field must be set to zero if the unused STU units cannot go back to the private account, or set to 1 if they can go back.

c=15 Modify the parameters associated with a user number, such as the division number or privileged status.



ss Error response field.

0	Parameter modification completed.
1	User not found.
2	Invalid subfunction.

sf Subfunction, specifying the parameter whose value is being changed with this message, as follows:

1	Snapback (where unused STU units are to go)
2	Division
3	Privilege/nonprivilege
4	Quantity of private STU units
6	Percentage of draw from the repository

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 8 of 16)

usrno — Binary user number of the user whose status is to be modified.

value — Value to be used for replacing the existing quantity. This field is zero or 1 if the value of the sf field is 1, 3, or 5; an ASCII division code if the value of sf is 2; in STU units if the value of sf is 4; or a binary integer between 0 and 100 if the value of sf is 6.

c=16 Suspend or resume execution of the task specified.

Beta(1)

ss	db	sf	flag	unused
8	8	8	8	32

ss — Error response field. The values are:

0	Function is successfully completed
1	Descriptor block is not assigned
2	Invalid descriptor block number
3	Descriptor block is not in a suspend state
4	Descriptor block is already suspended
5	Cannot suspend the operator task
6	Descriptor block is in a terminate state
7	Not enough CM space to resume task execution

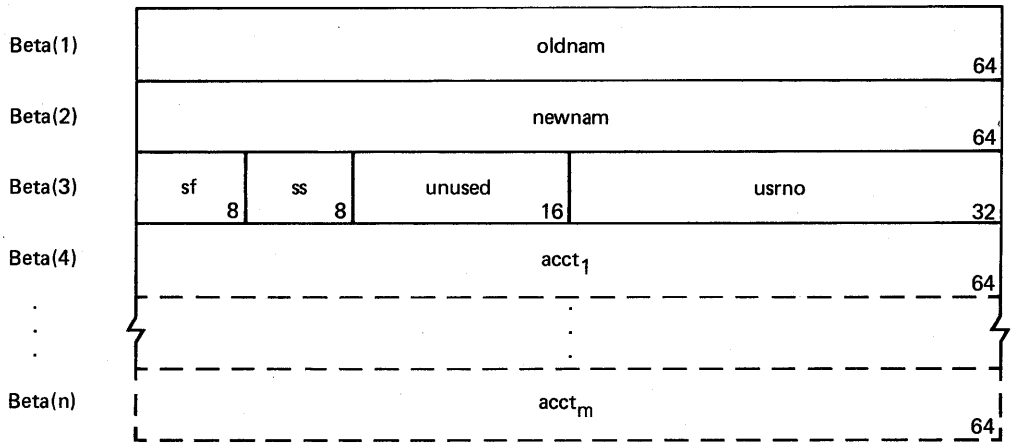
db — Descriptor block number of the task to be resumed or suspended.

flag — Valid only for sf=0. The values are:

0	Suspend task directly.
1	Write all pages and delete all pages except the minus pages from the page table.

sf — Subfunction. This field must be set to zero to suspend execution, or set to 1 to resume execution.

c=17 Modify an account, or list all accounts. When a list is returned, the operating system returns the length of the Beta portion in the n field of Alpha(2).



oldnam — Account number to be deleted or replaced; eight ASCII characters, left-justified with blank fill.

newnam — Account number to be created or to replace oldnam; eight ASCII characters, left-justified with blank fill.

sf — Subfunction. The values are:

1	Delete the account
2	Replace the account
3	Create a new account
4	List all accounts in the system

ss — Error response field, or an account count. This field is set to zero for a successfully completed function, set to 1 if the existing account could not be deleted or a new account could not be created, or set to the number of accounts listed when the sf field is 4.

usrno — User number for which the account is to be modified. This field must be zero to indicate all user numbers.

$acct_i$ — Account number of an account in the system; eight ASCII characters, left-justified with blank fill. Returned by the operating system only when the sf field is 4.

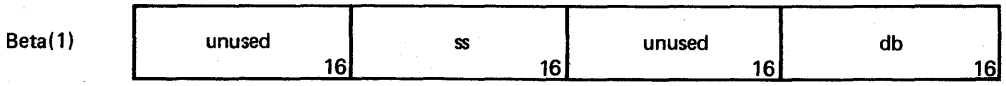
Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 9 of 16)

c=18 Get the operator's TTY number. The Beta portion consists of one word in which the operating system returns the binary TTY number (rightmost 16 bits only).

c=19 Turn output file processing on or off for all users. If output is turned off, no output files are transferred until output is turned on. The Beta portion consists of one word in which bits 1 through 56 are unused and bits 57 through 64 are the a field.

a=0 Turn output on
a=1 Turn output off

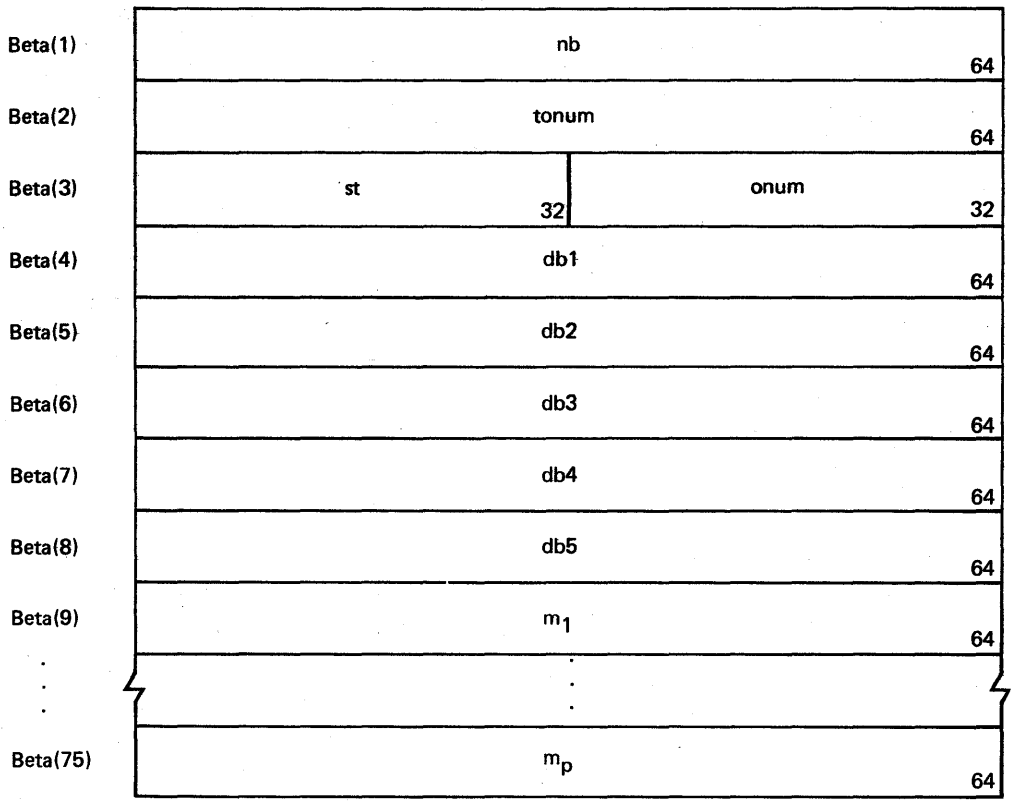
c=1A Abort the tape request.



ss Error response field. This field is set to zero for a tape request aborted, or set to 1 if the descriptor block is not assigned to the tape file.

db Descriptor block number of the tape requestor.

c=1B Display character strings to the operator. The operating system returns up to five retained strings. Only those strings that require a response by the operator, but which have not yet received a response, can be displayed.



nb Total length, in bytes, of the strings; set by the operating system. Maximum is 536.

tonum Total number of outstanding strings; set by the operating system.

st Oldest string in the system buffer to start with; set by the user. (Oldest string is identified as 1.) This field must be set to zero if the display is by descriptor block number.

onum Number of strings to be displayed, in binary; set by the user. Maximum number of strings is five.

dbi Descriptor block number of the originator of the string to be displayed, or else is unused; set by the user. Valid only when the st field is zero. Beta words Beta(4) through Beta(8) must always be reserved for DB numbers, although not all need be used.

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 10 of 16)

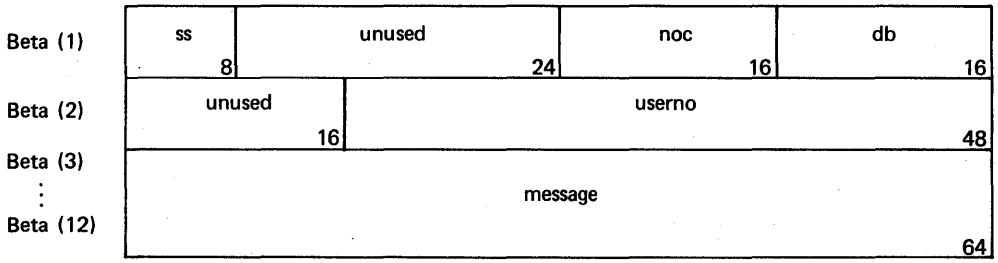
m_i Requested for strings, with 0D0A between each string, and terminated by a 1F. The strings are in the same format as for the SEND A MESSAGE TO CONTROLLER message (f=0014).

c=1C Checkpoint option. Set the checkpointed suffix for all the files belonging to the batch job in the file index. Write file index entries for all batch local files. Switch batch input files to checkpointed input files. Switch output files to checkpointed output files and give back to originating user. Save pool names. No Beta required.

c=1D Turn terminal logon processing on or off for all users except the operator. If terminal logon processing is turned off, no logon lines are accepted until terminal logon processing is turned on.

flag
- 0 Turn logon terminal processing off.
- 1 Turn logon terminal processing on.

c=#1E Record a comment line in a job's dayfile.



ss Error response code.
- 0 No error
- 1 Invalid descriptor block number/descriptor block free
- 2 Invalid binary user number/user not active
- 3 Not a batch job
- 4 Dayfile full
- 5 Non-existent dayfile
- 6 Dayfile not opened for implicit I/O
- 7 Invalid OPCOM call

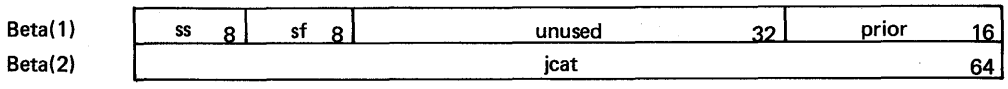
noc Number of characters in the message.

db Descriptor block number of the batch processor or controllee.

userno Binary user number.

message Message to be sent.

c=#1F Set a new maximum or default priority for the specified job category (CHPR operator commend).



ss Error response code.
- 0 No error.
- 1 Job category not found in system table.
- 2 Invalid parameter value.

sf Subfunction code indicating the priority type to be set.
- 0 Maximum priority.
- 1 Default priority.

prior Priority, 1 (lowest) through 15 (highest).

jcat Job category (8 ASCII characters, left-justified, blank-filled).

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format) (Sheet 11 of 16)

c=20 Evict a job from the input queue (EVIC operator command). This option purges the job file from the system.

Beta(1)	ss 8	unused	24	userno		32
Beta(2)	jobname					64

ss Error response code.

0 No error.
1 Job not found in input queue.
2 User not found in input queue.

userno Binary user number to which job belongs.

jobname Batch input file name (8 ASCII characters, left-justified, blank-filled).

c=21 Submit the specified job to the CPU scheduler immediately without regard to its priority or time limit (EXPD operator command).

The Beta format for this option is identical to the Beta format for the c=20 option.

c=22 Free the specified job, enabling its submission to the CPU scheduler (FREE operator command). This option can reverse the action of a c=23 option (HOLD operator command).

Beta(1)	ss 8	unused	24	userno		32
Beta(2)	jobname					64

ss Error response code.

0 No error.
1 Job name not found in input queue.
2 User number not found in input queue.

userno Binary user number to which job belongs or zero. Zero indicates all held jobs are freed.

jobname Batch input file name (8 ASCII characters, left-justified, blank-filled). This field is ignored if the userno field contains zero.

c=23 Hold the specified job in the input queue, preventing its submission to the CPU scheduler (HOLD operator command).

The Beta format for this option is identical to the Beta format for the c=20 option.

c=24 Turn on or turn off the submission of jobs for the specified job category (INPT operator command).

Beta(1)	ss 8	unused	48	sf	8	
Beta(2)	jcat					64

ss Error response code.

0 No error.
1 Job category does not exist.
2 Invalid subfunction (sf field value).

sf Subfunction code.

0 Turn input on for the specified job category.
1 Turn input off for the specified job category.

jcat Job category (8 ASCII characters, left-justified, blank-filled). If jcat is all blanks, the command applies to all job categories.

c=25 Set the maximum large page limit for any job within the specified job category (MXLP operator command).

Beta(1)	ss 8	unused	24	n		32
Beta(2)	jcat					64

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 12 of 16)

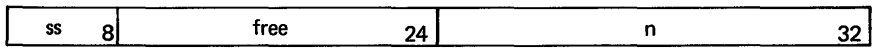
ss Error response code.

0 No error.
1 Job category does not exist.
2 Invalid parameter value.

n Maximum number of large pages allowed (0 through maximum maximum large pages in machine, MAXWS/128).

jcat Job category (8 ASCII characters, left-justified, blank-filled).

c=26 Set the percentage of memory overcommitment allowed (MXMO operator command).

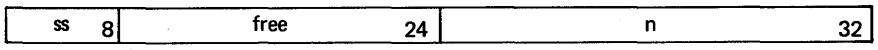


ss Error response code.

0 No error.
1 Invalid parameter value.

n Integer (0 through 10000) indicating allowed memory overcommitment. The integer is divided by 100 to obtain the percentage.

c=27 Set the maximum combined time limit for all jobs currently executing in the system (MXRR operator command).

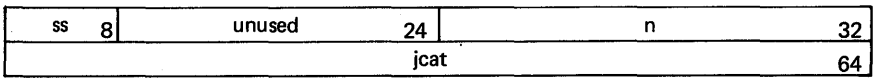


ss Error response code.

0 No error.
1 Invalid parameter value.

n Number of system minutes (1 through 9999).

c=28 Set the maximum time limit for any job within the specified job category (MXTL operator command).



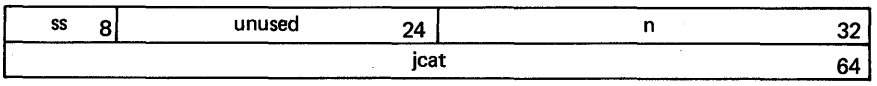
ss Error response code.

0 No error.
1 Job category does not exist.
2 Invalid parameter value.

n Time limit in system seconds (1 through 599 940).

jcat Job category (8 ASCII characters, left-justified, blank-filled).

c=29 Set the maximum working set size for any job within the specified job category (MXWS operator command).



ss Error response code.

0 No error.
1 Job category does not exist.
2 Invalid parameter value.

n Working set size limit in 512-word blocks (0 through the maximum blocks in machine, MAXWS).

jcat Job category (8 ASCII characters, left-justified, blank-filled).

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 13 of 16)

c=#2A Set the maximum number of executing jobs for the specified job category (SJCT operator command).

Beta(1)	ss	8	unused	24	n	32
Beta(2)	jcat	64

ss Error response code.

0	No error.
1	Job category does not exist.
2	Invalid parameter value.

n Maximum number of jobs that can concurrently execute (1 to 50).

jcat Job category (8 ASCII characters, left-justified, blank-filled).

c=#2B Create or delete job category validation for user (CREATE JOBCAT and DELETE JOBCAT operator commands).

Beta(1)	ss	8	ctype	8	unused	16	userid	32
Beta(2)	jcat	64

ss Error response code.

0	No error.
1	Job category does not exist.
2	User number does not exist.
3	Invalid command type (ctype field value).
4	Default job category (JDEFAULT) specified.

ctype Command type.

1	Create validation.
2	Delete validation.

userid Binary user number.

jcat Job category (8 ASCII characters, left-justified, blank-filled).

c=#2C Retrieve H display information for all input queue entries (refer to the CYBER 200 Operator's Guide for a description of the H display). The message returns values in all Beta fields.

For this option, the user must set the value of the bl field in Alpha(3). Following the input queue statistics, the system returns the job category mnemonics, the input queue order vector, and the input queue entries.

Beta(1)	ss	8	ctmo	16	ctrr	24	fnum	16
Beta(2)	mxmo	16	mxrr	24	maxws	24
Beta(3)	$jcat_1$	64
. . .	. . .
Beta(68)	$jcat_{66}$	64
Beta(69) . . . Beta(580)	T_IQV vector
Beta(581)	$finfo_1$	64
. . .	. . .
Beta(n)	$finfo_{fnum*5}$	64

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 14 of 16)

ss — Error response code.

0	No error.
1	More input queue entries exist than can fit in the reserved Beta area.

ctmo — Current percentage of memory overcommitment.

ctrr — Current number of system minutes required to rerun all executing jobs.

fnum — Number of input queue entries to return.

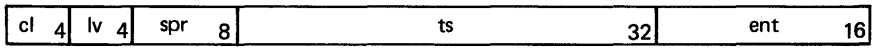
mxmo — Maximum percentage of memory overcommitment allowed.

mxrr — Maximum rerun time (in system minutes) allowed for all executing jobs.

maxws — Maximum working set size.

$jcat_i$ — Job category mnemonics, one mnemonic per word (ASCII left-justified, blank-filled). The site can define up to 66 job categories.

T_IQV — Input queue order vector, up to 512 words long, one word per input queue entry. The format for each vector word is as follows.



cl,lvl — Job class and level (determined by the job priority and the priority map table.

spr — Subpriority set by the input queue manager.

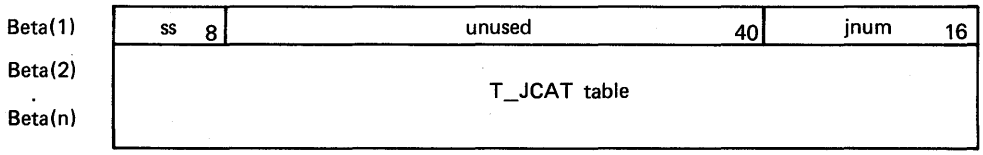
ts — Time stamp.

ent — Input queue entry number.

$finfo_i$ where i=1 to fnum*5 — Copies of the input queue entries (five words per entry). The format of an input queue entry is shown in figure 4-32.

c=#2D — Retrieve V display information for all job categories (refer to the CYBER 200 Operator's Guide for a description of the V display). The message returns values in all Beta fields.

For this option, the user sets the value of the bl field in Alpha(3). The system returns the information in the Beta.



ss — Error response code.

0	No error.
1	More entries exist than can fit in the Beta.

jnum — Length of job category table returned in Beta. (The entire table is returned.)

$jinfo_i$ — Copy of the T_JCAT system table (three words per job category). The format of the T_JCAT table is shown in figure 4-33.

c=#2E — Change the priority or time limit for the specified input queue job (STPR and STTL operator commands).

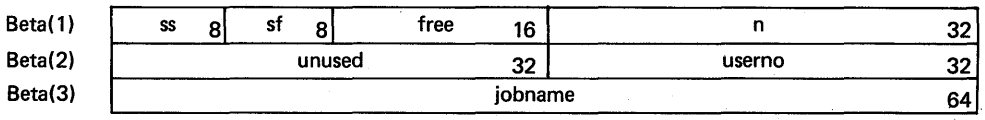


Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 15 of 16)

ss Error response code.

0	No error.
1	Job not found in queue.
2	User not found in queue.
3	Invalid parameter value.

sf Subfunction code.

0	Set new priority.
1	Set new time limit.

n New priority or time limit value (1 through the maximum allowed for the job category).

userno Binary user number to which job belongs.

jobname Batch input file name (8 ASCII characters, left-justified, blank-filled).

c=2F Retrieves job categories for which the specified user number is validated (DISPLAY JOBCAT operator command).

For this option, the user must set the bl field of the Alpha to the Beta length. The length must be not less than the number of valid job categories plus one.

Beta(1)	ss 8	jnum 8	unused 16	userno 32
Beta(2)	$jcat_1$ 64
Beta(jnum+1)	$jcat_{jnum}$ 64

ss Error response code.

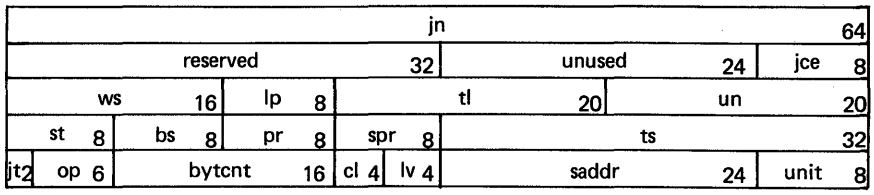
0	No errors.
1	Beta length specified in the bl field of the Alpha portion is insufficient for return of all valid job categories.
2	User number not found.

jnum In this field, the system returns the number of job categories it returned.

userno Binary user number whose valid job categories are returned.

$jcat_i$ Job category mnemonic (1 to 8 ASCII characters, left-justified, blank-filled).

Figure 4-31. EXECUTE OPERATOR COMMAND Message (Beta) Format (Sheet 16 of 16)



jn — Job name (8 ASCII characters, left-justified, blank-filled).

jce — Number of job category entry in the job category table.

ws — Working set size in blocks.

lp — Large page limit.

tl — Time limit in system seconds.

un — User number.

st — Job status within input queue.

0	Job has not yet been processed by input queue manager.
1	Job category turned off.
2	Maximum jobs running in job category.
3	Job held due to operator command.
4	Batch suffix busy.
5	Job exceeds maximum working set size.
6	Job exceeds maximum time limit.
7	Job exceeds maximum memory overcommitment.
8	Job exceeds maximum rerun time.
9	Job exceeds maximum large page limit.
10	No CM space available.
20	Job submitted to CPU scheduler.
40	Job has terminated.

bs — DB number of the highest level controller running at the batch suffix (set if the batch suffix is busy).

pr — Priority.

spr — Subpriority assigned when job entered the input queue.

ts — Time job entered the system (8 hexadecimal digits in the form ddhhmmss where dd is the day, hh is the hour, mm is the minute, and ss is the second).

jt — Job terminated field.

0	Job has not terminated.
1	Job has terminated.

bycnt — Bypass count; number of times input queue manager has bypassed the job.

op — Operator action on job.

0	No operator action.
1	Evicted job.
2	Expedited job.
3	Freed job.
4	Held job.

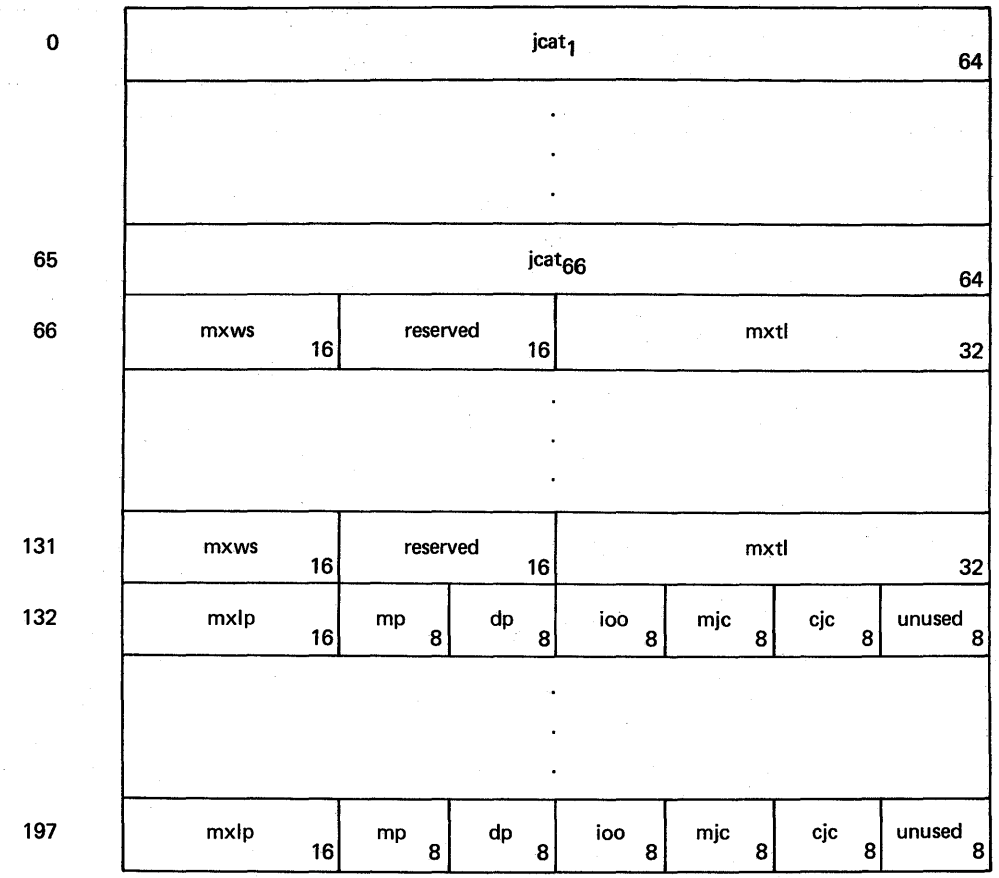
cl — Job class.

lv — Job level.

saddr — Beginning sector address of the job file.

unit — Unit number of the job file.

Figure 4-32. Input Queue Entry Format



jcat — Job category mnemonic (ASCII left-justified, blank-filled).

mxws — Maximum working set size limit in blocks.

mxtl — Maximum time limit in system seconds.

mxlp — Maximum large page limit.

mp — Maximum priority.

dp — Default priority.

ioo — Input on or off status.

0	Input on.
1	Input off.

mjc — Maximum job count; the maximum number of jobs from this category that can concurrently execute.

cjc — Current job count.

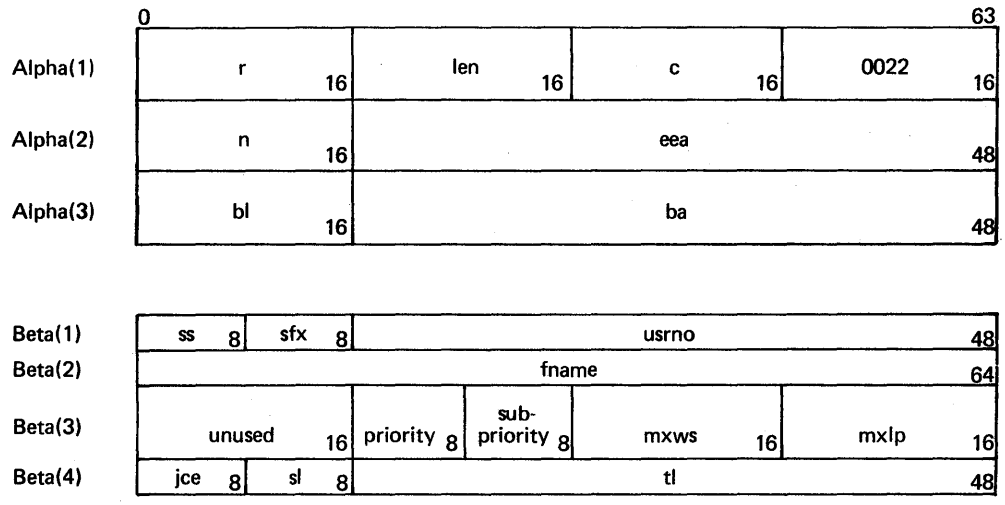
Figure 4-33. T_JCAT System Table Format

EXECUTE PROGRAM FOR USER NUMBER (f=0022)

The EXECUTE PROGRAM FOR USER NUMBER system message initiates execution of a file for a specified user number. Only a privileged user can issue this message. The initiated file can be local, attached permanent, attached pool, or public.

The USER1 routine IQM uses this message to initiate the batch processor.

The message format is shown in figure 4-34.



r — Response code returned by CYBER 200 OS when message processing is complete.

0	No errors.
2	File request error (refer to the ss field in Beta(1)).
3	No table space available.
4	Invalid message option (c field).
10	File index table, FILEI, is full.
214	Beta buffer length error. Either the first word address of Beta plus its length is greater than the maximum virtual user address, or the Beta buffer is too small for the number of requests and length specified.

len — Beta location.

#FFFF	Alpha(3) contains the length and virtual bit address of Beta.
Any other value	Beta begins at Alpha(3) (The Alpha and Beta portions of the message are contiguous.)

c — Message option.

1	Initiate file. (This is the only valid option).

n — Number of files put into execution with this message.

eea — Error exit address; virtual bit address to receive control if an error occurs during message processing (r=0). If this field is zero when an error occurs, the task is aborted.

bl,ba — If len = #FFFF, these fields contain the length in words and the virtual bit address of the first full word of the Beta portion.

ss — Error response code.

0	No errors.
1	No free suffix for this user number (sfx=0), or the suffix is busy for this user number (sfx=0).
2	User directory is not on the disk.
3	File was not found or not attached.
4	Nonexecutable file.
5	Invalid priority value.
6	Invalid user number.
7	Not authorized to run at priority level.
#B	Requested time limit exceeds time available to the user.

Figure 4-34. EXECUTE PROGRAM FOR USER Number (f=0022) (Sheet 1 of 2)

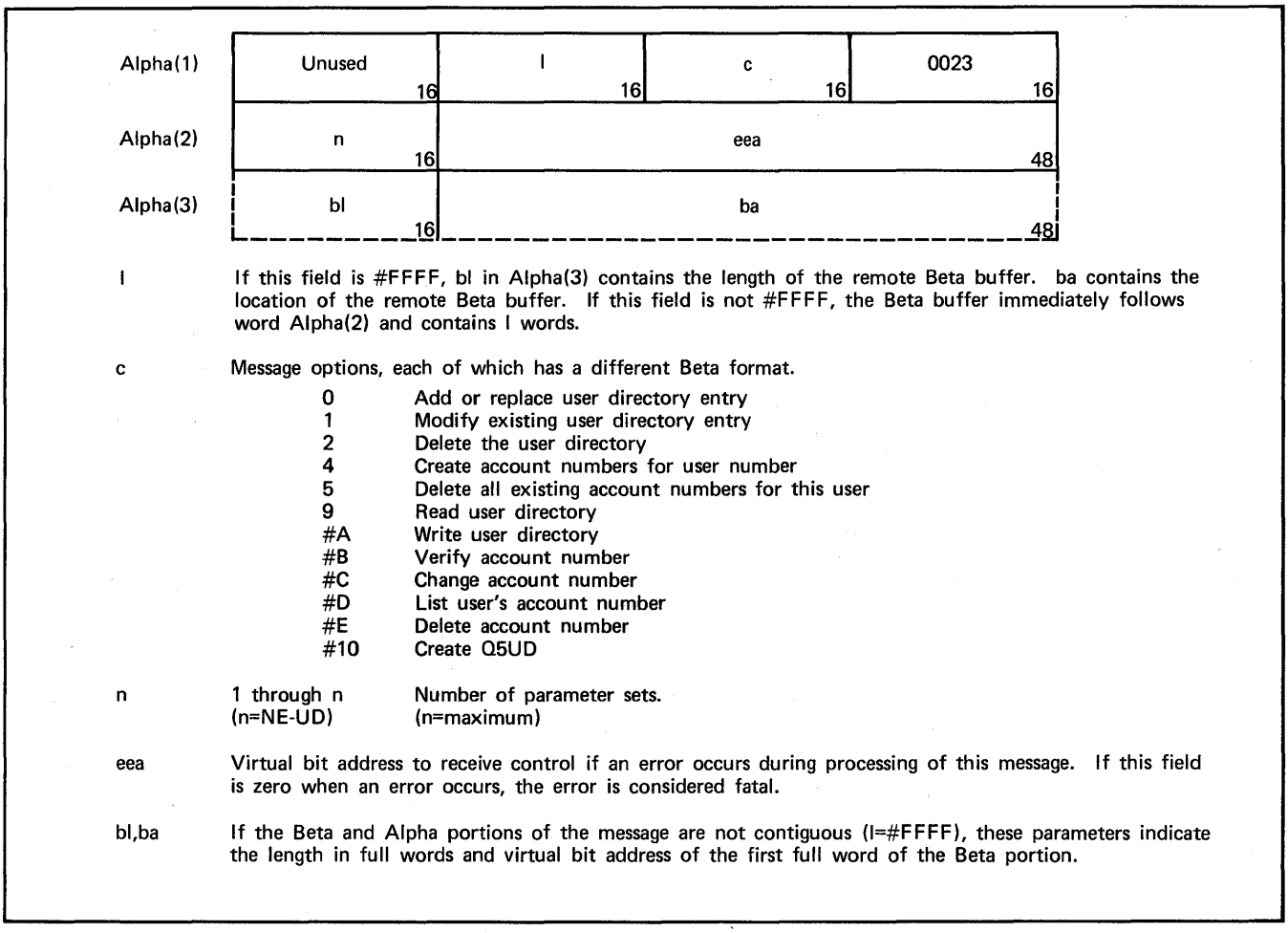
sfx	Suffix under which the task is to run. 0 CYBER 200 OS selects a free suffix and returns it in this field. 1 Suffix A. 2 Suffix B. 3 Suffix C. 4 Suffix D.
usrno	User number (6 ASCII characters, left-justified, blank-filled).
fname	File name (8 ASCII characters, left-justified, blank-filled).
priority	Job priority, 1 (lowest) through 15 (highest).
subpriority	Subpriority of the job (1 through 255).
mxws	Maximum working set limit in blocks (0 through the maximum blocks in machine, MAXWS in MISCTAB).
mxlp	Maximum large page limit (0 through the maximum large pages in machine, MAXWS/128).
jce	Job category entry number (0 through 65).
sl	Security level (ASCII character; default is P). S System access (level 7). A Accounting access (level 5). K Restricted access (level 3). P Unrestricted access (level 2).
tl	Time limit in system seconds (0 through 599 940).

Figure 4-34. EXECUTE PROGRAM FOR USER Number (f=0022) (Sheet 2 of 2)

UPDATE USER DIRECTORY (f=0023)

This message can be issued only by a privileged user and allows the user to create, delete, or modify a user directory, and to modify a division's repository account. One purpose for which an installation can use this message is to create a utility for managing batch job accounting (see section 7, Accounting).

The format of the Alpha portion of this message is shown in figure 4-35. The Beta word formats depend on the message option (c field) in Alpha(1), and are shown in figure 4-36.



l — If this field is #FFFF, bl in Alpha(3) contains the length of the remote Beta buffer. ba contains the location of the remote Beta buffer. If this field is not #FFFF, the Beta buffer immediately follows word Alpha(2) and contains l words.

c — Message options, each of which has a different Beta format.

0	Add or replace user directory entry
1	Modify existing user directory entry
2	Delete the user directory
4	Create account numbers for user number
5	Delete all existing account numbers for this user
9	Read user directory
#A	Write user directory
#B	Verify account number
#C	Change account number
#D	List user's account number
#E	Delete account number
#10	Create Q5UD

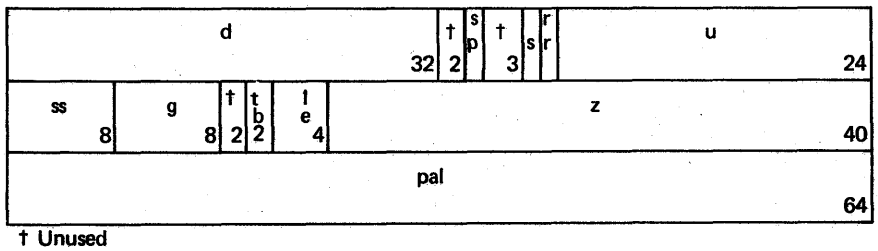
n — 1 through n (n=NE-UD) — Number of parameter sets. (n=maximum)

eea — Virtual bit address to receive control if an error occurs during processing of this message. If this field is zero when an error occurs, the error is considered fatal.

bl,ba — If the Beta and Alpha portions of the message are not contiguous (l=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-35. UPDATE USER DIRECTORY Message (Alpha) Format

c=0 Replace existing user directory or add a user directory if not already there.



† Unused

d Division code of the user, in ASCII, right-justified with blank fill.

sp This bit must be set to 0 if the user is not allowed to be the pool boss, or set to 1 if that user is allowed to be the pool boss.

s Unused time is returned to the repository if this bit is set to 0, or returned to a private account if this bit is set to 1.

rr Allows the user to run priority tasks. If this bit is set to 0, a priority task is not allowed; or, if set to 1, priority tasks are allowed.

u Binary user number.

ss Error response field. The values are:

1	Not a privileged user
2	User is active.
6	Percent is greater than 100

g Maximum percentage donation from the division's repository account allowed for this user (0 through #64).

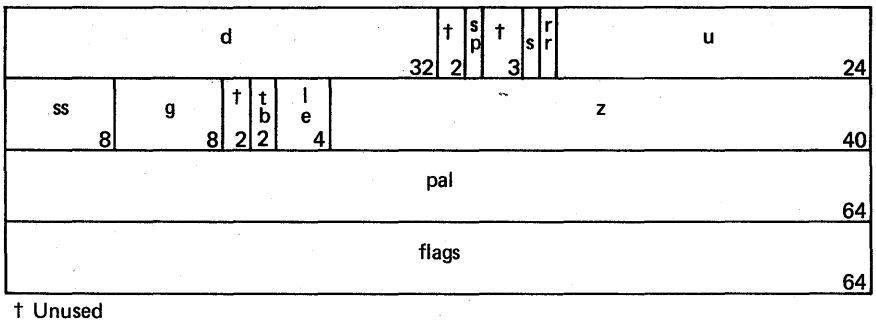
tb This field must be set to 0 if the user is not to be privileged, or set to 1 if the user is to be privileged.

le Maximum authorized access level (0 through 7).

z Bank account quantity to be installed in microseconds (referred to as money).

pal Public file access list.

c=1 Modify user directory/ specify fields (refer to c=0 for description of fields).



† Unused

ss Error response field.

#F	Not a user
2	User is active

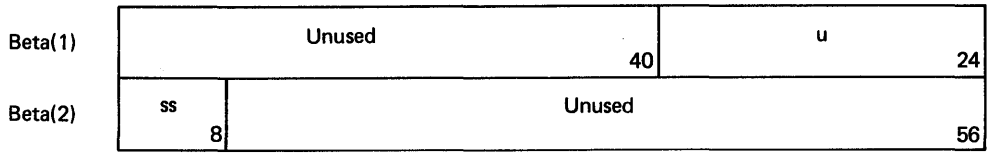
Figure 4-36. UPDATE USER DIRECTORY Message (Beta) Format (Sheet 1 of 5)

flags Flags corresponding to fields requiring modification.

1	Modify
0	Don't modify

Bit Position	Corresponding Field	Bit Position	Corresponding Field
0	d	13	z
1	unused	14	pal
2	unused	15-63	unused
3	sp		
4	unused		
5	unused		
6	unused		
7	s		
8	rr		
9	g		
10	unused		
11	tb		
12	le		

c=2 Delete this user. Two Beta words are required

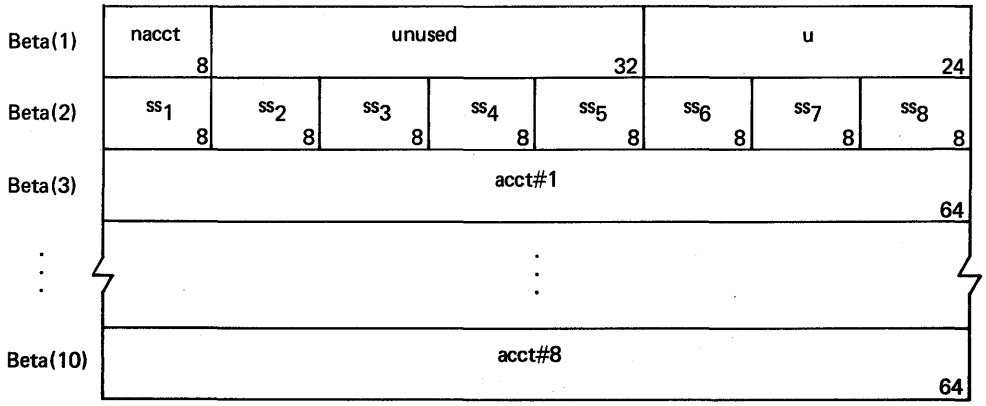


u User's binary user number to be deleted.

ss Virtual system error code.

0	Acceptable response
2	User is active
#F	Not a user

c=4 Add up to eight account numbers for a user. The total number of account numbers added cannot exceed NE_UDACCT.



nacct Number of accounts to add to user.

u Binary user number.

ss_1-ss_8 Error code process Acct #1 - #8

9	Account number exists
#A	Account file overflow
#F	Not a user

acct#1 First ASCII account number to be added (left-justified, blank filled).

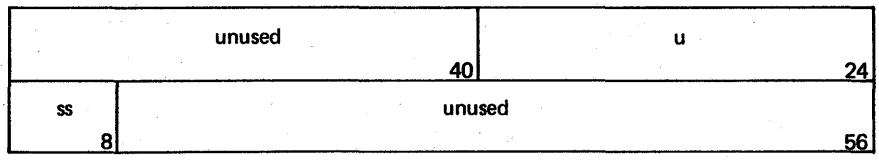
.
.
.

acct#8 Eighth ASCII account number to be added (left-justified, blank filled).

NE_UDACCT Number of accounts allowed per user.

Figure 4-36. UPDATE USER DIRECTORY Message (Beta) Format (Sheet 2 of 5)

c=5 Delete all account numbers for this user.

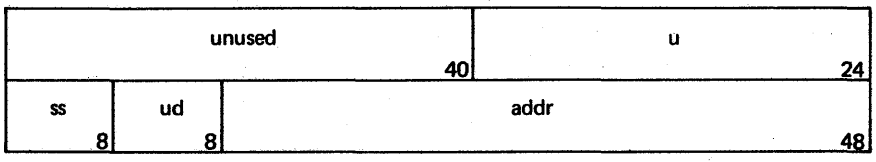


u Binary user number for account numbers to be deleted.

ss Error code return.

0	Acceptable response
#F	Not a user

c=9 Read user directory.



u Binary user number.

ss Error response field.

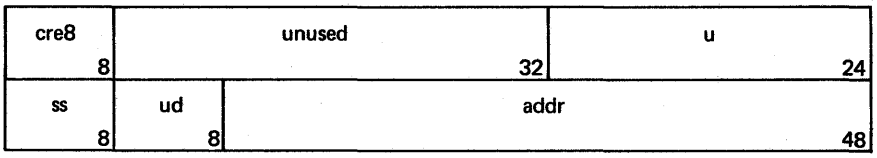
0	Acceptable response
#F	Not a user

ud User directory.

0	Read 5 words of User Table information
1	Read entire User Directory entry

addr Virtual address from which UD is to be read

c=#A Write user directory/ or CREATE user.



cre8 If equal to #7F, create user
If not equal to #7F, write user directory

u Binary user number.

ss Error response field.

0	Acceptable response
2	User is active
#E	User already exists (cre8=#7F)
#F	Not a user (cre8≠#7F)
#10	U does not match user at addr

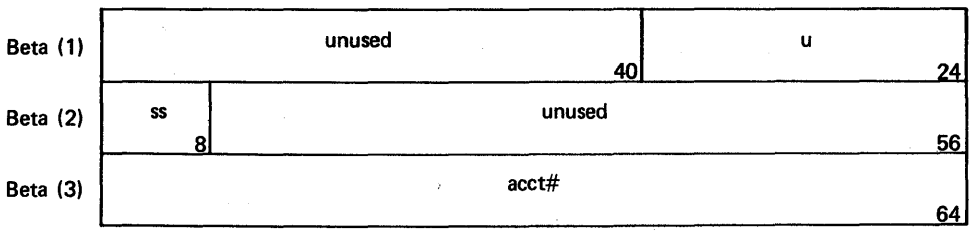
ud User directory.

0	Write five words of user table information
1	Write entire user directory entry

addr Virtual bit address from which user directory is to be written.

Figure 4-36. UPDATE USER DIRECTORY Message (Beta) Format (Sheet 3 of 5)

c=#B Verify account number.



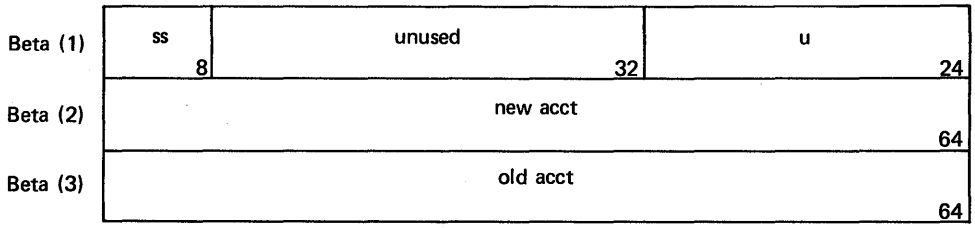
u Binary user number.

ss Error response field.

0	Acceptable response
#B	Account number does not exist
#F	Not a user

acct# ASCII account number to be verified (left-justified, blank filled).

c=#C Change account number.



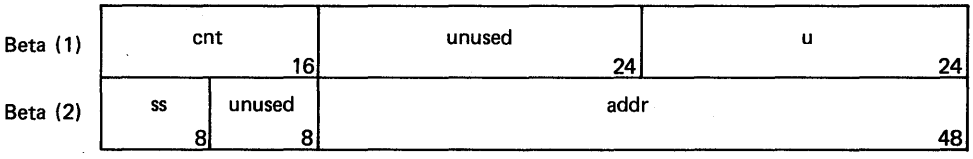
ss Error response field, as defined for c=#B.

u Binary user number.
#FFFFFF = Change account number for all users.

new acct New account number in ASCII (left-justified, blank filled).

old acct Old account number in ASCII (left-justified, blank filled).

c=#D Read user account numbers.



cnt Number of accounts read (returned by operating system).

u Binary user number.

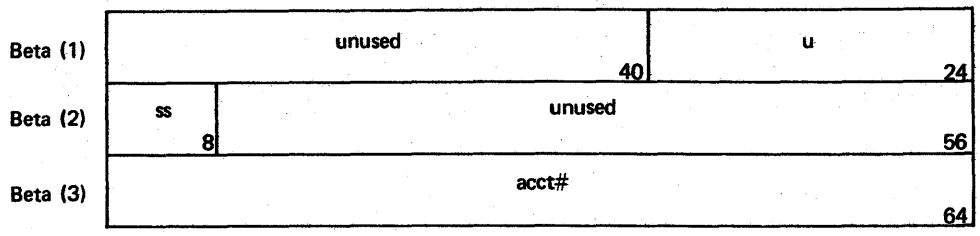
ss Error response field.

0	Acceptable response
#F	Not a user

addr Accounts are to be read in at this location.

Figure 4-36. UPDATE USER DIRECTORY Message (Beta) Format (Sheet 4 of 5)

c=#E Delete one account number.



u Binary user number.
#FFFFFF = Delete account number for all users.

ss Error response filed, as defined for c=#B.

acct# ASCII account number to delete (left-justified, blank filled).

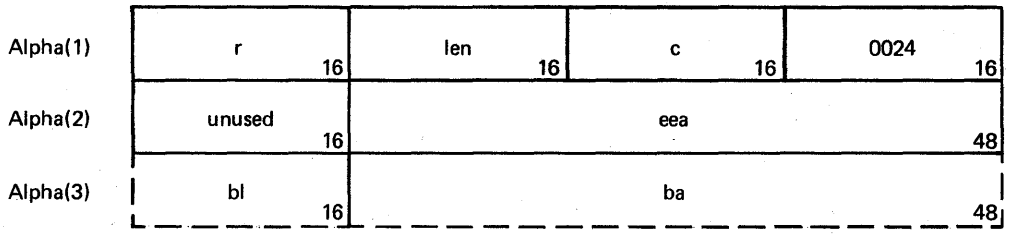
c=#10 Initialize Q5UD file. Beta is not required.

Figure 4-36. UPDATE USER DIRECTORY Message (Beta) Format (Sheet 5 of 5)

MISCELLANEOUS (f=0024)

With this message, a user program can determine a variety of information concerning itself, its controller, and its controllees. Also, raw accounting statistics can be retrieved with option 9 (c field).

The format of this message is shown in figure 4-37. The Beta portion of the format is discussed under the c field description.



r Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	No controllee, illegal information option, or the error response field for Beta(1) of option value C, 12, or 13 is set
2	Program not currently running in interrupt program state (c=#F or #11 only).
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. For option 8, if this field is 3, Beta(4) is not present.

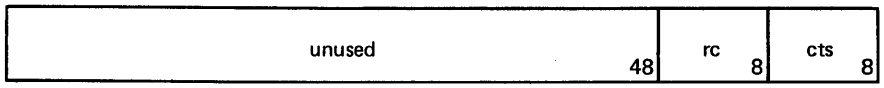
c Message options (c=2 and c=4 are reserved). All information is returned by the operating system unless otherwise specified. The values are:

Figure 4-37. MISCELLANEOUS Message Format (Sheet 1 of 4)

1 Get the user number and bank account. Beta(1) contains the user number in ASCII, right-justified with blank fill, in the rightmost 48 bits; the leftmost 16 bits are unused. Beta(2) contains the quantity of STU units in the user's bank account, in the rightmost 40 bits; the leftmost 24 bits are unused.

3 Get the time limit. Beta(1) contains the existing time limit, in microseconds, in the rightmost 48 bits; the leftmost 16 bits are unused.

5 Get the controllee's termination state. Beta(1) contains the following:



rc Controllee's return code. rc=0 means successful completion; other values are:

- 4 Nonfatal error
- 8 Fatal error

cts Controllee's termination state. The values are:

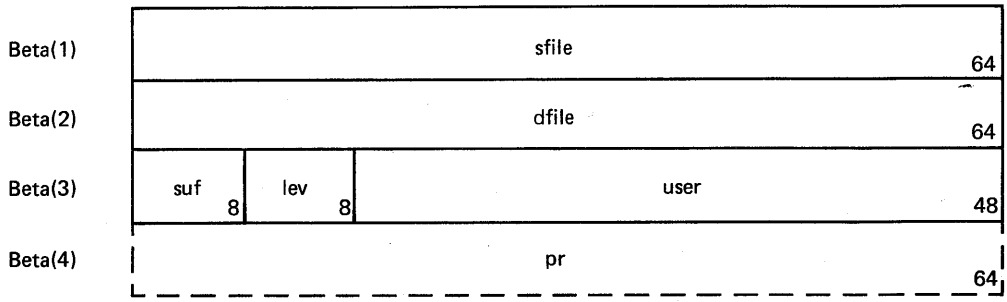
- 0 Still active
- 1 User terminal break to the EXIT control statement
- 2 Operator break to the EXIT control statement
- 3 Operator break to the end-of-job (6/7/9) card
- 4 Operator drop of the entire batch deck
- 39 Normal termination
- 3D Abort
- 3E Normal termination

The drop, scratch, and output files are saved for 3D and 3E. For 39, the drop and scratch files are destroyed, while the output file is given to an output processor according to the disposition code for the file.

6 Get the controllee's name and place. Beta(1) contains the source file name for the controllee, and Beta(2) contains the drop file name for the controllee.

7 Get the controller's name and place. Beta(1) contains the source file name for the controller, and Beta(2) contains the drop file name for the controller.

8 Get this program's name and place. The Beta portion of the format is:



sfile Source file name for this program.

dfile Drop file name for this program.

suf Indicates the program's suffix (as specified in the LOGON line):

- 41 Suffix A
- 42 Suffix B
- 43 Suffix C
- 44 Suffix D

lev Level in the controllee chain; $1 \leq lev \leq 9$.

user User number in ASCII, right-justified with blank fill.

pr Identifies the privileged task. This field must be set to zero for a nonprivileged task, or nonzero for a privileged task. Unless the len field in Alpha(1) is 4, this Beta word does not exist.

Figure 4-37. MISCELLANEOUS Message Format (Sheet 2 of 4)

9 Get the raw page fault counts, CPU times, and memory usage. If the batch processor issues this option, the statistics returned include the cumulative statistics for the batch processor and all its controllees. The Beta portion of the format is as follows:

Beta(1)	spflt 32	lpflt 32
Beta(2)	unused 16	ucpu 48
Beta(3)	unused 16	memu 48
Beta(4)	unused 16	syscpu 48

spflt Number of disk accesses (input requests issued) that resulted from small page faults (small page implicit reads) up until the issuance of the MISCELLANEOUS message.

lpflt Number of disk accesses (input requests issued) that resulted from large page faults (large page implicit reads) up until the issuance of the MISCELLANEOUS message.

ucpu User execution CPU time, in microseconds, up until the issuance of the MISCELLANEOUS message.

memu Memory usage; the values (current working set size)*(user CPU time for the current accounting period) summed over all accounting periods for the task up until the issuance of the MISCELLANEOUS message.

syscpu Virtual and resident system CPU time, in microseconds, for user execution up until the issuance of the MISCELLANEOUS message.

A Get clock times as of message issuance. The Beta portion of the format is as follows:

Beta(1) Master clock value, expressed as yymmddhhmmssppppp. pppp is fraction of a second.

Beta(2) ASCII clock value, expressed as hh.mm.ss (hour.minutes.seconds).

Beta(3) Calendar value, expressed as mm/dd/yy (month/day/year).

Beta(4) Value of the millisecond clock (0 at midnight).

Beta(5) Value of the microsecond central processor clock (0 at power on).

Beta(6) Current date, in the rightmost 16 bits. The leftmost seven bits of the 16 bits contain the last two digits of the year, in binary; the remaining nine bits contain the number of days since the beginning of the year (1 to 366), in binary. The leftmost 48 bits of Beta(6) are unused.

B Reserved.

C Get the time for the repository and add it to the user's account. Beta(1) contains the return code in the leftmost 16 bits; the values are zero for a successful addition, 1 for no repository access, and 2 for not enough time available. The leftmost 48 bits of Beta(1) are unused. Beta(2) contains the time value, in floating point form, representing the number of minutes to be obtained from the repository. If the return code is 2, the maximum time available for use is returned by the operating system in Beta(2).

D Get the contents of the minus pages and return them in Beta(1) through Beta(1024). (The format of the minus pages is described in section 2.) If there is no second minus page, Beta(513) is #FFFF. If beta length is 512, only the first minus page is returned.

E Get the version identifiers. Beta(1) contains the resident system version identifier, and Beta(2) contains the virtual system version identifier.

F Get the interrupt invisible package. If the program is running in interrupt state, Beta(1) through Beta(40) contain the contents of the invisible package saved when a program interrupt occurred. (See appendix E, which describes the invisible package.) If the program is not currently running in interrupt state, a response of 2 is returned.

10 Get the task CPU time. Beta(1) contains the task CPU time, in microseconds.

11 Get the interrupted register file. Beta(1) through Beta(256) contain the contents of registers 0 through 255, when a program interrupt occurred. If the program is not currently running in the interrupt program state, a response code of 2 is returned.

12 If the batch input file whose name is supplied by the user in Beta(1) fails to complete due to a system failure, destroy the batch job's input file. The name of the batch input file must be left-

Figure 4-37. MISCELLANEOUS Message Format (Sheet 3 of 4)

justified with blank fill. Beta(2) contains the return code; the value is zero if the batch job's input file is successfully destroyed, or 1 if the batch file names does not exist.

13 If the batch input file whose name is supplied by the user in Beta(1) fails to complete due to a system failure, rerun the batch input file. The name of the batch input file must be left-justified with blank fill. Beta(2) contains the return code; the value is zero if the batch input file is successfully rerun, or 1 if the batch file name does not exist.

eea Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

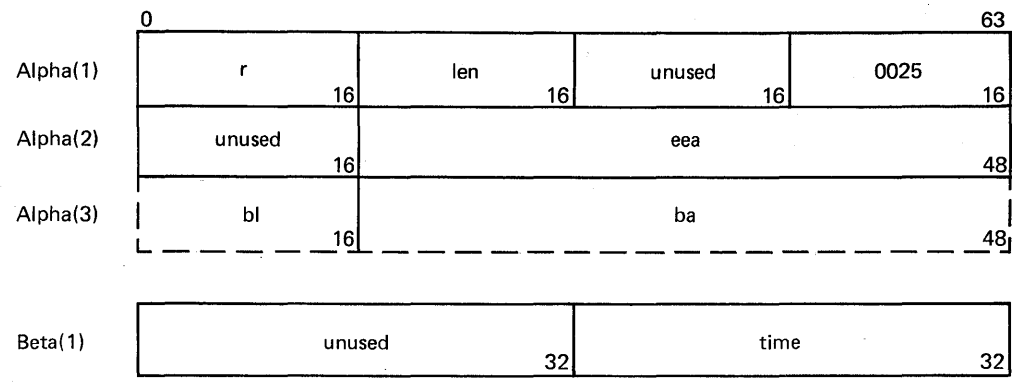
Figure 4-37. MISCELLANEOUS Message Format (Sheet 4 of 4)

RECALL (f=0025)

The RECALL message allows a program to suspend its own execution for not fewer than 30 seconds nor more than 30 minutes. At the end of suspension, the program is recalled to an active status. The format of this message is shown in figure 4-38.

POOL FILE MANAGER (f=0026)

This message offers a variety of options relating to pool file management, including attaching the user to a specified pool (thus giving the user access to the files in the pool) and detaching a pool (after which the files in the pool are no longer accessible to that user). A user who issues option 1 of this message to create a pool becomes that pool's pool boss. Only the pool boss can issue options 2, 3, 6, and 7. Only the pool boss or a user authorized by the pool boss can issue options 4 and 5. Any user can issue option 8.



r Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1 This message is not allowed for this program
214 Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion, that is, 1.

eea Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

time Period of suspension; specified as a hexadecimal integer number of microseconds. No fewer than 30 seconds and no more than 30 minutes must be specified by this field. Values outside this range are set to the nearest interval limit.

Figure 4-38. RECALL Message Format

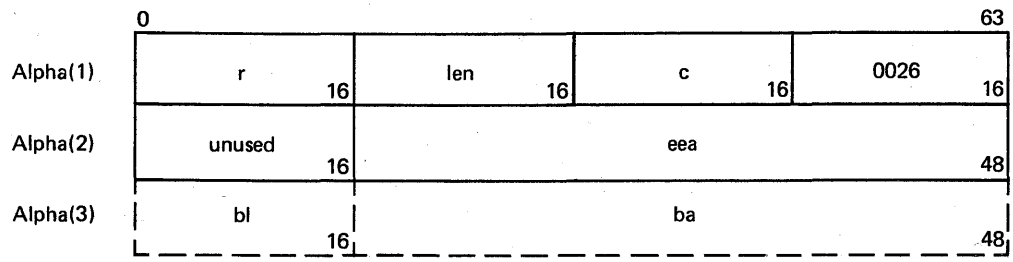
At the end of each batch job, at the time that the batch processor issues the USER/ACCOUNTING COMMUNICATION message option 2 (end-of-job), any pools that were first attached by the job are detached. Pools that have been attached interactively remain attached until the user detaches them or until the user is no longer active (that is, has done a (sc)BYE with no task still running). Hence, if the user interactively attaches a pool first, and also attaches the same pool later in a batch job (under a different suffix), the pool is not detached at the end of the batch job. Pools that have been attached by a batch job can be accessed interactively only for the duration of the batch job. Also, for any particular user number, a task running under one suffix can explicitly detach a pool that was attached under another suffix.

The format of the POOL FILE MANAGER message is shown in figure 4-39. The Beta formats are described under the c field definition.

LINK (f=0027)

With this message, a privileged user can perform various operations relating to the CYBER 200 Link Station. It can link or delink a subchannel number, add or delete station IDs, log-in or log-out the mainframe, or process a CYBER interactive output message.

The Alpha portion of the LINK Message is shown in figure 4-40. The Beta word formats depend on the message option (c field) in Alpha(1).



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero. A nonzero response code does not necessarily mean an error has occured (see message options 7 and 8). Other values are:

11	Pool name was already attached by this user
12	Pool name is undefined
13	Already in four pools
14	Pool is not attached
15	Cannot attach to the pool; user has no access to the pool
16	Undefined user number
17	Duplicate pool name
18	Unable to destroy the pool
19	Pool access directory (PAD) is full
1A	Pool list (PLIST) table is full
1B	Invalid pool
1C	Invalid pool name
1D	Not a pool boss
1E	PAD or PLIST file was not found (see the CYBER 200 Operating System Installation Handbook)
1F	File index table is full
214	Beta buffer or user list buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer or user list buffer is too small for the number of results and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion. If this field is 1 under message option 3, all users can access the pool.

c — Message options. The pool name field in Beta(1) contains up to eight letters and numbers and must start with a letter; it is left-justified with blank fill. Binary user numbers can range from 1 to 999999; they are right-justified with zero fill. The options are:

1	Create a pool. Adds the pool name to the pool list and clears (zeros) the pool access directory (invalid when universal access is set) for that pool. The creator is the pool boss. Beta(1) contains the pool name.
2	Destroy the pool. If no users are attached and no files are in the pool, the pool name is deleted from the pool list. Beta(1) contains the pool name.
3	Grant access to the pool. Places the specified user numbers into the pool access directory. If either the len field or bl field is 1, all users can access the pool. Beta(1) contains the pool name. Beta(2) contains the length and address of the user number list buffer.

Figure 4-39. POOL FILE MANAGER Message Format (Sheet 1 of 2)

Beta(2)

length (16)	address (48)

length — Length of user number list buffer (in words).

address — Bit address of user number list buffer.

4 Attach to the pool. Attaches the requestor to the named file pool. Beta(1) contains the pool name.

5 Detach the requestor from the pool. Beta(1) contains the pool name.

6 Remove the access privilege. Specified user numbers are removed from the pool access directory. Beta(1) contains the pool name. Beta(2) contains the length and address of the user number list buffer.

7 List the users having access to the pool. Beta(1) contains the pool name. Beta(2) contains the length and address of the user number list buffer.

8 List the pools and the pool boss. All nonzero entries in the pool list file are copied into a variable number of Beta words; the number of words copied is returned in the response code field. Each pool list entry returned by the operating system has the following format:

Beta(1)	pcount (16)	pptr (16)	pfree (12)	pboss (20)
Beta(2)	poolnm (64)			

pcount — Count of users attached to this pool.

pptr — Pointer into the pool access directory (PAD) that contains a list of user numbers which can attach to this pool (the value is used by the operating system only).

pfree — Currently unused.

pboss — User number of the boss of this pool.

poolnm — Name of this pool, in ASCII.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion. Under option 3, all users can access the pool if the bl field is 1.

Figure 4-39. POOL FILE MANAGER Message Format (Sheet 2 of 2)

Alpha(1)	r (16)	len (16)	c (16)	0027 (16)
Alpha(2)	n (16)	eea (48)		
Alpha(3)	bl (16)	ba (48)		

r Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Illegal option
2	The message is for privileged use only
201	No link file table entry exists for this db/zip
211	Number of files was zero
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified
301	CYBERIN cannot be started at this time; SMP should respond with a file ordinal equal to zero to this message
501	No entry exists for this physical station identifier
D01	Mainframe tables full or logical identifiers are exhausted
D02	The zip and pid fields do not match the mainframe table
D03	Logical identifiers are exhausted
E01	Could not locate the physical identifier in the mainframe table
1001	Mainframe table is full
1002	Mainframe is already logged in
1101	Zip is zero cannot logout CYBER 200
1202	Could not locate the db/zip in the link file table

len If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length of the Beta portion.

c Message options:

1	(not used)
2	Update the link file table according to the status
3	Spool the input file request
4	(not used)
5	List the station identifier table
6	(not used)
7	(not used)
8	(not used)
9	(not used)
A	(not used)
B	(not used)
C	(not used)
D	Add the station identifiers
E	Delete the station identifiers
F	(not used)
10	Log-in the mainframe
11	Log-out the mainframe
12	Enter the dayfile message
13	(not used)
14	(not used)
15	(not used)
16	Poll - A list of logged on terminals. The system returns the identifiers of terminals that have outstanding output along with the output messages.

n For all options, this field is the size of the Beta portion of the message, and the Beta area should be at least n words long. When the message has been processed, n is set equal to the word length of the information returned in the Beta portion for options 5 and 16.

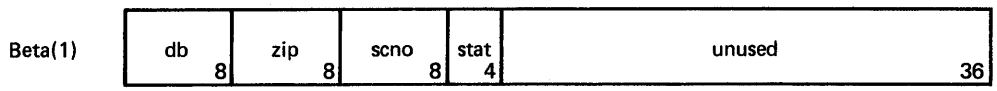
eea Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length in full words and virtual bit address of the first full word of the Beta portion.

Figure 4-40. LINK Message (Alpha) Format (Sheet 1 of 4)

c=1 Not used.

c=2 Update the link file table according to the status. The caller must set the db, jip, and stat fields. For link-up (stat=3) scno must also be set. Only one entry can be modified per LINK call. Beta(2) is reserved for file delink. For stat=2, the parameter package is returned beginning in Beta(2).



db Descriptor block number of link file.

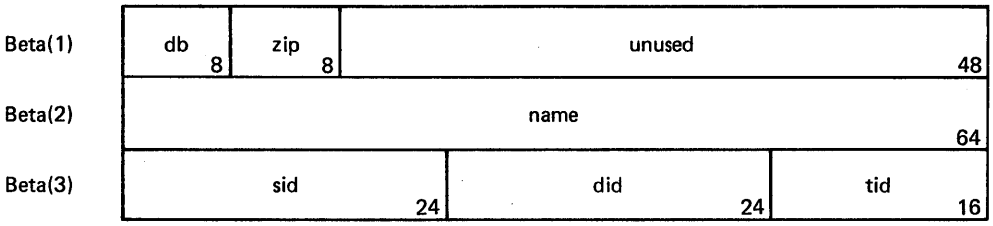
zip Zip code associated with this request.

scno Subchannel number.

stat Status. Possible values:

1	Create outstanding.
2	File information.
3	File linkup.
4	Abort.
5	Abort outstanding.
6	Abort in progress.
7	Delink or abort.

c=3 Spool input file request.



db Descriptor block number of the spool input file; returned by the operating system when the spool input request can be processed.

zip Zip code associated with this request.

name Name of the spool input file.

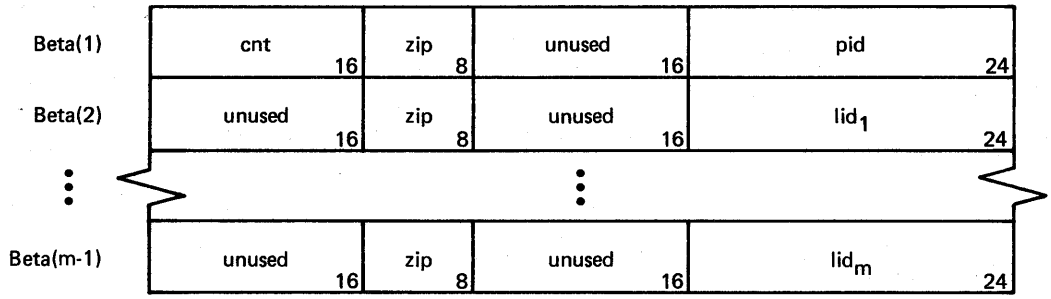
sid Originating terminal.

did Destination terminal.

tid Terminal identifier.

c=4 Not used.

c=5 Station identification table.



cnt Number of logical identifiers.

pid Physical identifier.

zip Zip code for this pid/lid; returned by the operating system.

lid_i Logical identifiers.

Figure 4-40. LINK Message (Alpha) Format (Sheet 2 of 4)

This represents one station entry. All stations, including CYBER 200, are returned if the pid field in Beta(1) is zero. If the pid field is set, only the entry for this physical identifier is returned.

c=6 Not used.

c=7 Not used.

c=8 Not used.

c=9 Not used.

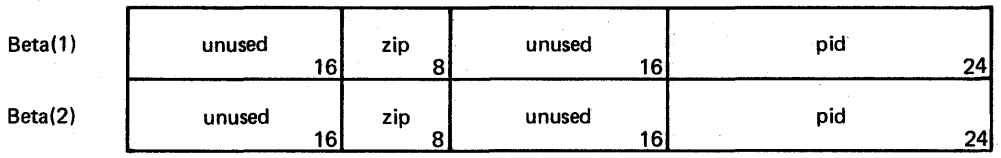
c=A Not used.

c=B Not used.

c=C Not used.

c=D Add the station identifiers.

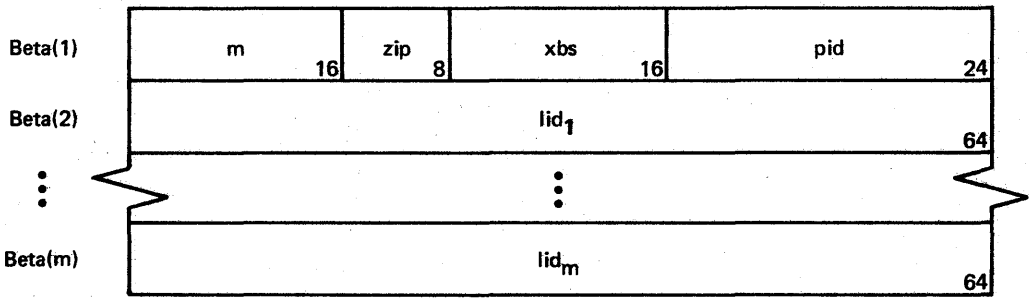
c=E Delete the station identifiers.



zip Zip code associated with the physical identifier in the pid field. The zip field must be set for a linked mainframe; if this field is zero, it means the CYBER 200 mainframe.
pid Three-character mainframe physical identifier.
lid Three-character logical identifier associated with the physical identifier in the pid field.

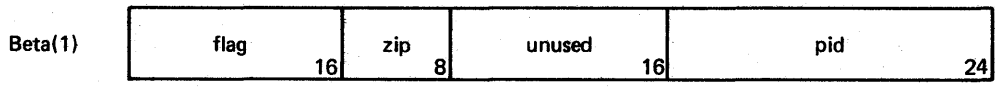
c=F Not used.

c=10 Log-in the mainframe.



m Number of logical identifiers to be added.
zip Zip code associated with the physical identifier in the pid field. The zip field must be set.
xbs Maximum transmission block size. This field is set to the front-end transmission block size by the caller, or set to the minimum of the front-end and CYBER 200 transmission block size by the operating system.
pid Three-character mainframe physical identifier.
lid_i Three-character logical identifier associated with the physical identifier in the pid field.

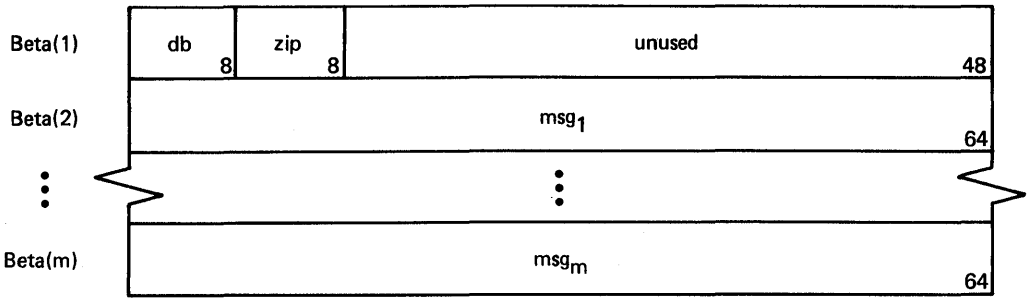
c=11 Log-out the mainframe.



flag Nonzero if link is logged out. Zero if link is brought up.
zip Zip code associated with the physical identifier in the pid field. This field must be set.
pid Three-character mainframe physical identifier.

c=12 Enter the dayfile message.

Figure 4-40. LINK Message (Alpha) Format (Sheet 3 of 4)



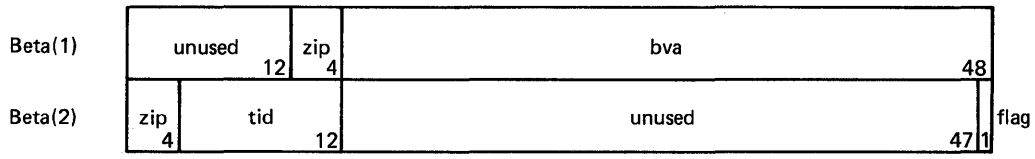
db Descriptor block number.
zip Zip code associated with this request.
msg_i Dayfile message, in ASCII.

c=13 Not used.

c=14 Not used.

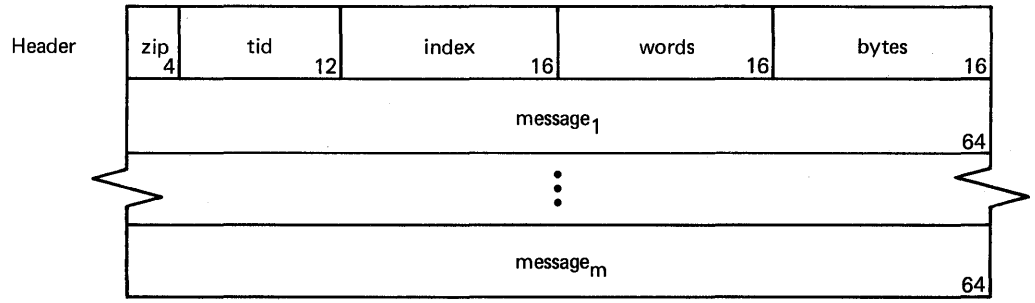
c=15 Not used.

c=16 Poll. SMP sets its zip code in Beta(1). SMP puts the terminals logged into CYBER beginning in Beta(2). The number of terminals is put in the n field of Alpha(2), by the caller. The length of the output in words is returned in the n field of Alpha(2) by the operating system.



zip in Beta(1) Equivalent to zip in Beta(2).
bva The system returns output messages at this bit address.
zip in Beta(2) Zip code associated with the front-end Link Station.
tid Two 6-bit display code characters, representing the terminal id associated with the front-end Link Station.
flag Set if Link Station cannot accept any output for this tid at this time. The system does not return output (if any) for this terminal.

Format for Output Messages:



zip Zip code associated with front-end.
tid Two 6-bit display code characters representing terminal id of front-end Link Station.
index Current byte pointer in message.
words Number of words including header.
bytes Number of bytes in the message.
message Message text.

Figure 4-40. LINK Message (Alpha) Format (Sheet 4 of 4)

VARIABLE RATE ACCOUNTING (f=0028)

This message dynamically changes the variable rate during execution of a task. This call can be made only by a public controllee or a controllee which has the variable rate permit flag set in the User Directory. Dynamic calls to change variable rates are made by applications programs rather than utilities.

If IP_F_VR is zero, the call is illegal (r=2).

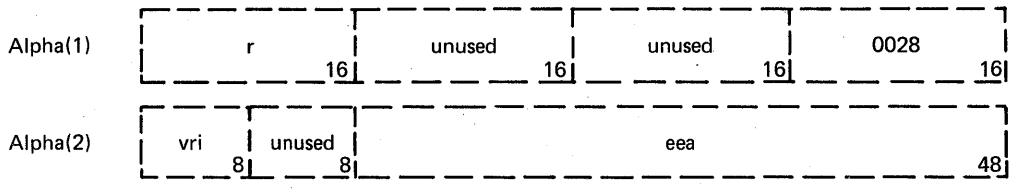
The change to the variable rate index is made in the descriptor block entry. At the time the change is made, BANKAC is called to compute the accumulated SBUs to be charged at the old rate and to decrement the available time remaining to complete the task.

Figure 4-41 shows the format of the Alpha portion of the variable rate accounting message.

SEND MESSAGE TO DAYFILE (f=0029)

This message allows the user to send a string of ASCII data to the program dayfile.

The format of the message is shown in figure 4-42.



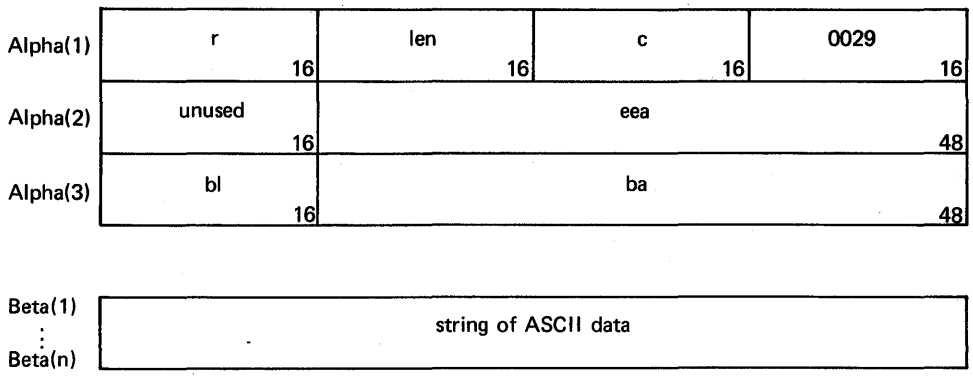
r Response code returned by the operating system when this message has been processed. If no error occurs, a response code of zero is returned; otherwise:

- r=1 File is not public and call is not validated for this account
- r=2 Call is not valid at this installation

vri Index into variable rate table (set by user).

eea Virtual bit address to receive control if an error occurs during message processing (if r is different from 0). If eea=0, the error is considered fatal.

Figure 4-41. VARIABLE RATE ACCOUNTING Message



r Response code; returned by the operating system when the message has been processed. If no error occurs, the error code is zero; otherwise:

1	Length of Beta in character bytes is greater than 4096
2	Illegal c field selected for this message
3	Dayfile is full. The message in dayfile will be DAYFILE FULL
4	Dayfile is not opened for implicit I/O
5	Invalid base virtual bit address for Q5DAYFILE
6	Unable to find Q5DAYFILE
7	The call is currently blocked waiting for access to dayfile (caller will not see a response code 7 when control is returned)
214	Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the message, length is zero.

len If len=#FFFF, Alpha(3) contains the length in character bytes and the virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha(3), and len is the length in character bytes of the Beta portion.

c Control field.

1	Add time stamp to message

eea Virtual bit address to receive control if an error occurs during message processing (r≠0). If eea is zero, the error is considered fatal.

bl,ba If the Beta portion of the message is not contiguous to the Alpha portion (len=#FFFF), these parameters indicate the length in character bytes and virtual bit address of the first full word of the Beta portion.

string of ASCII data
Maximum length of 4096 characters. If there is no #1F at the end of the line one will be added. Illegal characters #00 through #1E and #7F through #FF will be changed to blanks. The combination #0D0A will be changed to #201F.

Figure 4-42. SEND MESSAGE TO DAYFILE Message Format

EXECUTE IQM REQUEST (f=0030)

The EXECUTE IQM REQUEST system message processes input queue manager (IQM) requests. Only the input queue manager can issue this message.

The format of the EXECUTE IQM REQUEST system message is shown in figure 4-43.

Alpha(1)	r 16		len 16		c 16	0030 16	
Alpha(2)	n 16		eea 48				
Alpha(3)	bl 16		ba 48				

r — Response code returned by CYBER 200 OS when message processing is complete.

0	No errors.
1	An error occurred (refer to the ss field in the Beta(1)).
2	User number of message issuer is not that of the input queue manager.
3	Job file was not added to the input queue because the queue is full.
4	Invalid message option (c field value).
211	Either zero or too many Beta entries were specified.
214	Beta buffer length error. Either the first word address of Beta plus its length is greater than the maximum virtual user address, or the Beta buffer is too small for the number of requests and length specified.

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3) and len is the length in words of the Beta portion.

c — Message option.

1	Remove an entry from the input queue.
2	Add an entry to the input queue.
3	Clear input queue flags.

n — Number of input queue entries to remove (also number of Beta entries) (c=1 only).

eea — Error exit address; virtual bit address to receive control if an error occurs during message processing (r=0). If this field is zero when an error occurs, the task is aborted.

bl,ba — If the Alpha and Beta portions of the message are not contiguous (len = #FFFF), these parameters indicate the length in words of Beta and the virtual bit address of its first full word.

c=1 — Remove an entry from the input queue and reorder the subpriorities of the remaining entries as needed.

Beta(1)	unused 32	iqent 32	

iqent — Index of input queue entry to be deleted (0 through 511).

c=2 — Add an entry to the input queue. This option includes checking for valid job category, setting default values, setting the qf field of the file index entry to one, and setting the subpriority of the new entry.

Beta(1)	jn 64									
Beta(2)	jc 64									
Beta(3)	ws 16		lp 8		tl 20		un 20			
Beta(4)	ss 8	unused 8	pr 8	unused 8	saddr 24		unit 8			
Beta(5)	unused 32				time stamp 32					

jn — Job name (8 ASCII characters, left-justified, blank-filled).

jc — Job category (8 ASCII characters, left-justified, blank-filled).

ws — Working set size in blocks (0 through the maximum blocks in machine, MAXWS in MISCTAB). Zero sets the ws field to the default value of 500 blocks. If the ws field is equal to K_WS_MAX (#FFFF), the working set limit is set to the size of the machine, and the machine size is returned in the ws field.

lp — Large page limit (0 through the maximum large pages in machine, MAXWS/128).

tl — Time limit in system seconds (0 through 599 940). Zero indicates the default, ten seconds.

Figure 4-43. EXECUTE IQM REQUEST (f=0030) (Sheet 1 of 2)

un — Binary user number.

ss — Error response code.

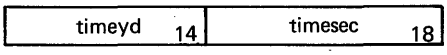
0	No errors.
1	Job category specified does not exist.
2	Invalid working set size. The MAXWS value from the MISCTAB table is returned in the ws field.
3	Invalid large page limit. MAXWS/128 is returned in the lp field.
4	Invalid time limit.
5	Invalid priority.
6	User locked out of specified job category.
7	Invalid user number.

pr — Priority. If this field is zero, use the default priority for this job category. If the specified priority is greater than the maximum priority for this job category, the priority is reset to the maximum for this job category. The priority is returned in the pr field.

saddr — Beginning sector address of the job file.

unit — Logical unit number of the device on which the file resides.

time stamp — Date and time when the file was created in the following format.

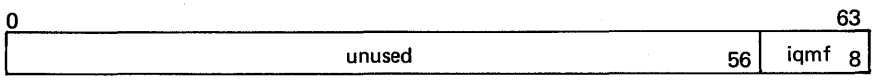


timeyd — Binary value resulting from the summation of the last two digits of the year and the ordinal of the day within the year (1 through 366).

timesec — Number of seconds since midnight.

c=3 — Clear the input queue flags.

Beta(1)



iqmf — Input queue manager flags indicating the reason for input queue processing. If a bit in iqmf is set, the corresponding bit in the input queue manager miscellaneous table is cleared to zero. The bit meanings are as follows:

Bit	Meaning
60	Interactive job terminated and a job in the input queue is waiting for CM space.
61	Batch job terminated.
62	Operator entered an IQM command.
63	USER1 routine gave a file to the input queue manager.

Figure 4-43. Execute IQM Request (f=0030) (Sheet 2 of 2)

EXPLICIT INPUT/OUTPUT (f=0050)

This message allows a user program to explicitly transfer data to and from buffers defined by the message. Explicit input/output is the only way that input/output can be performed with magnetic tapes. Input and output on mass storage files can be performed either explicitly (using this message) or implicitly (with read and write statements). Double buffering is easily accommodated by this message.

As soon as an EXPLICIT I/O message has been issued, program processing continues. When the message is set up properly, the user is notified when the input/output operations have been completed. By placing an interrupt request in the first pair of words in the Beta portion of the message and initializing the int field of the appropriate requests, the user program can specify that an interrupt occur on either successful or unsuccessful completion of an input or output request. Interrupt routines cannot be interrupted; they are stacked in the interrupt address stack and processed one at a time.

The Beta portion of the message can contain any selection of buffer definition requests, data transfer or function requests, and an optional interrupt request. The user can issue from one to eight input/output requests with each message. Up to six EXPLICIT I/O messages can be outstanding for a program at any one time; but if six requests are outstanding when a seventh request is issued, the issuing program is put in a wait program state (the operating system uses an INITIALIZE OR DISCONNECT CONTROLLEE message, f=001B) until at least one message has finished processing.

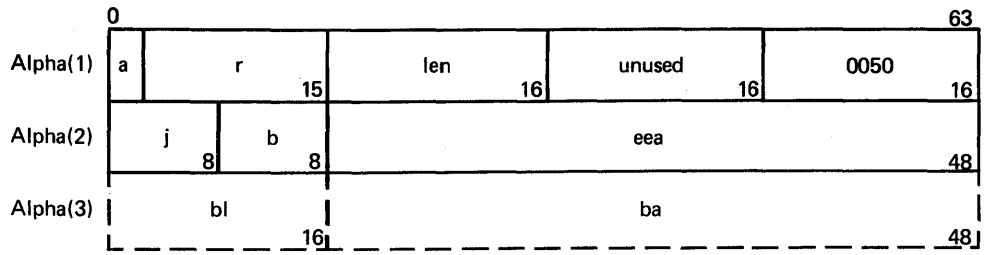
Within a message, requests can be made contingent on other requests; because requests are issued sequentially, the contingency must be upon previously issued requests. Interrupt upon completion of a request, therefore, can be contingent on the completion of other requests. The operating system is responsible for checking all possibilities in determining whether the interrupt condition has been met.

The program's minus page contains an input/output connector for each file the user program has opened; a file's input/output connector number is used to designate the file on which input/output is being performed. To perform explicit input or output on a file, the program must first open the file and define the input/output buffers. (When implicit input or output is being performed, the operating system must define the input/output buffers.) A buffer can be closed as soon as the data transfer request is issued; input/output need not first be completed. When a file is closed, all buffers are automatically closed that are defined for the file.

The Alpha and Beta words of the EXPLICIT I/O message should not be modified until all input and output described by the call is completed. To help the user determine when input/output is done, the a field in Alpha(1) has been defined; the user sets the bit before the message is issued, and the operating system clears the bit when the Alpha and Beta words are no longer of use.

When the central operating system detects an error before a request is sent to the peripheral operating system, the cerr field of the EXPLICIT I/O message is filled appropriately, control passes to the error exit address, and message processing terminates. A data transfer error detected by the peripheral operating system does not cause control to pass to the error exit address; however, the response code field is set to 4 and the serr field of the EXPLICIT I/O message is filled appropriately.

The format of the Alpha portion of the EXPLICIT I/O message is shown in figure 4-44. The formats of the Beta portion are shown in figure 4-45. The error codes appropriate for certain operations of the cerr field are shown in table 4-3.



a — This bit is cleared by the operating system to indicate to the caller when the Alpha and Beta words are no longer required. The caller should set this bit if this feature is to be utilized.

r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Illegal interrupt address
2	More than eight requests
4	Error occurred in one or more requests
214	Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin as Alpha(3), and len is the length in words of the Beta portion. Up to eight requests, which would result in a Beta length of 16, are allowed for each explicit input/output message.

j — Index, indicating which Beta entry was last processed or is currently in process; returned by the operating system.

b — Index, indicating which Beta entry caused the interrupt; returned by the operating system.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

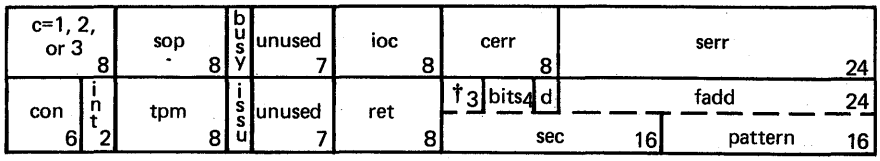
bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

Figure 4-44. EXPLICIT INPUT/OUTPUT Message (Alpha) Format

c Message option (operation to be performed):

1 Read
2 Write
3 Function specified in the sop field
4 Buffer operation specified in the sop field
5 Interrupt routine specification

c=1 Read data from a tape or disk file to a buffer. A buffer definition request must precede this request in the Beta portion.



†Unused

sop Suboperation to be performed. The suboperation depends on the value of the c field. When the c field is 1 or 2, the values are:

1 Buffer 1
2 Buffer 2

When the c field is 3, the values are:

3 Rewind
4 Unload (tape only)
5 Write an end-of-file marker
6 Space forward n tape records; the fadd field must contain the value of n
7 Read to the end of the file
8 Backspace n tape records; the fadd field must contain the value of n
9 Backspace to the beginning of the previous file
A Set tape density; the fadd field must contain 0, 1, 2, or 3 for 200 bpi, 556 bpi, 800 bpi, or 1600 bpi
C Erase tape or mass storage; the fadd field must contain the number of 6-inch portions of tape to be erased, for tape; the sec and pattern fields in Beta(2) must be given values, for mass storage
F Read status; the unit status is returned in the fadd field, as follows:

Bit	Meaning	Bit	Meaning
12	Reserved	18	End of tape
13	Parity error	19	Load point
14	End of operation	20	End of file
15	Data lost	21	Write enable
16	800 bpi density	22	Busy
17	556 bpi density	23	Ready
16 and 17	1600 bpi density		

If bits 16 through 17 are zero, 200 bpi density is indicated. Bits 0 through 11 are always zero.

When the c field is 4, the values are:

1 Open buffer 1
2 Close buffer 1
3 Open buffer 2
4 Close buffer 2

When the c field is 5, the sop field is not used.

busy If the caller has set this bit, it is cleared when the request is completed.

ioc Input/output connector number for the file on which input and output are being performed.

Figure 4-45. EXPLICIT INPUT/OUTPUT Message (Beta) Format (Sheet 1 of 4)

cerr Errors detected by the central operating system before the request is sent to the peripheral system. The values are:

1	Nonexistent input/output connector
2	Buffer size is greater than 24 small pages, or is zero pages for a small page buffer definition request
3	This file is not on physical mass storage
4	Illegal tape density requested
5	Illegal operation or suboperation
6	Illegal value in the tpm or mode field
7	No buffer was assigned
8	File address is out-of-bounds
9	Illegal attempt to access a read-only or write-only file
A	Interrupt requested, but no interrupt address was specified
B	Request for greater than 128 small pages (for a large page buffer definition request)
C	Buffer crosses a large page boundary (for a large page buffer)
D	Read or write access was denied for this buffer
E	Attempt was made to open a buffer that was already open
F	Buffer is already in use; previous input/output, which uses the same buffer, is not complete
10	Attempt to reuse Alpha before the previous call, which uses the same Alpha address, is complete
11	Reserved for CYBER 200 Link Station input/output (subchannel has aborted)
12	Attempt to read or write in an unassigned virtual space (buffer error)
13	Reserved for CYBER 200 Link Station input/output (mainframe logged out)

See table 4-2 for the error codes appropriate for certain operations.

serr Errors detected by the peripheral operating system. Multiple errors are possible and are all signaled. If the serr field is nonzero for any of the conditions listed, an interrupt occurs if the interrupt-on-error option of the int field has been selected. Bits are numbered in the field from left to right, 0 to 23. The bits and their meanings are:

Bit	Meaning
0 to 10	(no meaning)
11	Mainframe logged out (link only)
12	Error encountered in extending the file; end-of-file error has also been returned
13	Request aborted by the station operator
14	Error in positioning a mass storage device
15	Attempted backspace at the load point
16	Lost data from a tape record
17	Channel failed (disk only)
18	Attempted to write on a file-protected tape
19	Reached end-of-file; for a mass storage read or write, indicates that the buffer extends past the end of the file
20	Reached the end of the tape
21	Data quantity exceeds the user's buffer size
22	Tape, station buffer unit, or transmission parity error
23	Device is not ready

The serr bits for parity error, end-of-file, channel failure, and positioning mass storage are appropriate for mass storage input/output.

For all tape input/output operations except status, the serr bits for tape not ready (23) and parity error (22) are appropriate. For read or write tape, the serr bits might be set for data exceeds buffer (21), end-of-tape (20), end-of-file (read only), and file-protected tape (18). For write end-of-file on tape and erase tape operations, the end-of-tape and file-protected serr bits might be set. For forward-space tape record and forwardspace tape file, the end-of-tape and end-of-file serr bits might be set. For backspace tape record and backspace tape file, the end-of-file and load point serr bits might be set. The hba field in the ioc and file index is not updated for an erase.

con Contingency field; specifies how to handle the requests in this message. If this field is zero, there is no contingency; proceed immediately to the next request. If it is 3, there is a contingency; wait for completion of this request before processing the next.

The caller becomes a candidate for central processor use once the system encounters a data transfer or function request having the con field set to 3, or when no more Beta word pairs exist. Users of the EXPLICIT I/O message are responsible for detecting data transfer completion by selecting an interrupt condition, checking when the busy bit in the Beta word pair is clear, or issuing a GIVE UP CPU UNTIL I/O COMPLETES message (f=0052). When issuing multiple betas with con=3, errors detected by the central system cause control to pass to the error exit address only in the first beta. For subsequent betas, the user must check the error fields after completion to determine if an error occurred.

Figure 4-45. EXPLICIT INPUT/OUTPUT Message (Beta) Format (Sheet 2 of 4)

int — Interrupt conditions. This field has meaning only if an interrupt Beta request (c=5) is the first request in this message. The values are:

1	Interrupt on satisfactory completion of the request
2	Interrupt if an error occurs on this request
3	Interrupt on either condition (1 or 2)

tpm — Tape mode:

0	External binary-coded decimal (7-track tape), even parity
1	Unformatted binary (7-track or 9-track tape), odd parity
2	6-bit ASCII (7-track tape), even parity

issu — A bit set by the operating system when the central operating system finds no error, and a request is sent to the peripheral system.

ret — Retry field:

0	Use standard error recovery
1	No retry on error
2	Use standard error recovery and system noise records (tapes)
3	No retry on error and system noise records (tapes)

bits — When reading tapes not generated on CYBER 200 systems, this field indicates the number of bits read from tape (fewer than 16), in addition to the byte count returned in the fadd field. When reading CYBER 200 generated tapes, if this field is nonzero, the byte count in the fadd field must be decremented by one to indicate the amount of valid data read from tape.

For read link, the number of valid data bits returned in addition to the last full 16-bit byte. For write link, the number of valid data bits to be transmitted after the last full 16-bit byte.

d — Transmission directions. If this field is zero, truncate the record to the 64-bit word boundary. This field must be set to 1 to transmit an entire record.

fadd — The value in this field depends on the values in the c and sop fields. For mass storage read or write operations (c=1, c=2), fadd contains the small page address where data transmission is to begin. For magnetic tape read or write operations, fadd contains the number of 16-bit bytes to be transmitted to or from the buffer; if fadd is zero, the entire buffer is transmitted. Following the read or write operation, fadd contains the number of 16-bit bytes in the tape record (for tape), or the actual number of small pages transmitted (for mass storage).

For various subfunction operations (c=3), fadd contains a density selection (sop=A), count of records to space forward or backward (sop=6, sop=8), amount of erasure (sop=C), or a read status (returned by the operating system when the sop field is F). When an end-of-file is encountered on a backspace or forwardspace operation, the operation stops and the actual number of records spaced over is returned in fadd.

For mass storage, logical page (block) address where data transmission is to begin; for magnetic tape, the number of 16-bit bytes to be transmitted (if zero, entire buffer is transmitted). When op=3 and sop=A, fadd is used for density selection. When op=3 and sop=6 or 8, fadd must contain the count of records to skip. For read link, the number of 16-bit bytes transmitted to the buffer. For write link, the number of 16-bit bytes to be transmitted over the link (if zero, the entire buffer is transmitted).

sec — Sector count; erase mass storage functions only (c=3, sop=C, ioc = the number of an input/output connector for a file on mass storage). If the sector count given is zero, the operating system erases the entire file.

pattern — Bit pattern to be written; erase mass storage functions only (c=3, sop=C, ioc = the number of an input/output connector for a file on mass storage).

c=2 — Write data from a buffer to a tape or disk file. A buffer definition request must precede this request in the Beta portion. The Beta words for this option are the same as for c=1.

c=3 — Perform the subfunction specified in the sop field. The Beta words for this option are the same as for c=1.

c=4 — Open or close a buffer. Up to two buffers can be defined at one time; one buffer is defined by one request. A buffer can be closed as soon as the data transfer request is issued. Input/output need not be completed.

Figure 4-45. EXPLICIT INPUT/OUTPUT Message (Beta) Format (Sheet 3 of 4)

Beta(1)

c=4	sop	busy	unused	ioc	cerr	unused
8	8		7	8	8	24

Beta(2)

blen	unused	badd
8	16	40

sop — Suboperation to be performed. The suboperation depends on the value of the c field. The values are the same as for c=1.

ioc — Input/output connector number for the file on which input and output are being performed using this buffer.

cerr — Errors detected by the central operating system before the request is sent to the peripheral system. The values in this field are the same as for c=1.

blen — Length of the virtual range, in small pages, to be associated with this buffer. For mass storage and magnetic tape input/output, unless the buffer is in a large page the maximum buffer size is 24 small pages. The maximum large page buffer size is 128 small pages. A large page buffer must lie on a large page boundary and must not extend over the end of the large page.

If a record transmitted to the buffer is larger than the buffer, only as much data as fits in the buffer is transmitted; the rest of the record is lost. If a record transmitted to the buffer is smaller than the buffer, the remainder of the buffer is undefined.

badd — Starting virtual page address of the buffer where data transfer requests will deposit or obtain information.

c=5 Define where control is to be transferred if an interrupt condition occurs. If these Beta words are used, they must be the first ones in the Beta portion, and the int field of another request in this Beta portion must be initialized. If Beta(2) is nonzero, Beta(2) is placed in register 1E when control passes to the interrupt address, ia.

Beta(1)

c=5	unused	ia
8	8	48

Beta(2)

dbl	dba
16	48

ia — Virtual bit address where control transfers upon interrupt (such as an end-of-tape operation).

dbl — Length of the data base to be established if an interrupt occurs.

dba — Address of the data base to be established if an interrupt occurs.

Figure 4-45. EXPLICIT INPUT/OUTPUT Message (Beta) Format (Sheet 4 of 4)

TABLE 4-3. APPROPRIATE ERROR CODES FOR INPUT/OUTPUT OPERATIONS

File Type	c Field	cerr Field																	
		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
Mass storage	1	x	x			x	x	x	x	x	x	x	x	x		x	x		x
	2	x	x			x	x	x	x	x	x	x	x	x		x	x		x
	3	x				x	x			x	x						x		
	4	x		x		x									x		x		
	5																x		
Magnetic tape	1	x	x			x	x	x			x	x	x	x		x	x		x
	2	x	x			x	x	x			x	x	x	x		x	x		x
	3	x			x	x	x				x						x		
	4	x				x									x		x		
	5																x		

An x indicates a valid type-c-cerr combination.

RETURN FROM INTERRUPT (f=0051)

For control to return to the calling routine, an interrupt routine must issue this message when it has finished performing its tasks for either an input/output or program message interrupt. The message consists of an Alpha portion only, which is shown in figure 4-46.

Because the interrupt routine (level 1) cannot be interrupted by any other software interrupts, it will run until it issues a RETURN FROM INTERRUPT message. The current interrupt is then released and its invisible package is lost. The level-0 invisible package becomes current, and its register file image is restored by the operating system. All information from the level-1 register file is lost.

When interrupts occur and the interrupt routine is already in control, the operating system stacks the interrupt information in the interrupt address stack in the program's minus page. When the interrupt routine issues a RETURN FROM INTERRUPT message, level 0 is not restarted. Instead, the next interrupt on the stack is taken. This process is repeated until the stack is empty.

An option in this message allows level 1 to become the new level 0 after all additional interrupts stacked for this and any other level-1 routines have been processed. In this case, the register file image for level 0 is lost at the time level 1 becomes level 0. The new level 0 can have its own level-1 interrupt routines.

GIVE UP CPU UNTIL I/O COMPLETES (f=0052)

A user program can issue this message to give up the central processor (suspend execution) until all or part of the program's input and output is complete. In this way, processing of other programs is not delayed while the user's program is waiting for input/output. This message is used in conjunction with the EXPLICIT I/O message (f=0050).

The operating system keeps a record only of outstanding input and output messages. If the address in Beta does not match an outstanding input or output message, the message is at that time presumed to be completed.

Stacked interrupts are handled differently for interrupt routines and routines at level 0. If an interrupt routine (a program at level 1) gives up the central processor until all input and output completes (n=0 or n=1), the interrupt routine is restarted immediately, when all input/output has completed; stacked interrupts are processed only after this interrupt routine issues a RETURN FROM INTERRUPT message (f=0051). If a program at level 0 gives up the central processor until all input and output completes (n=0), the level-0 program is restarted only after all input/output is complete and all stacked interrupts have been processed.

The format of this message is shown in figure 4-47. The addresses of more than one EXPLICIT I/O message can be specified in the Beta portion, one word per address.

ADVISE (f=FF00)

The ADVISE message is used by a program to inform the operating system of an anticipated change in the need for virtual space. The ADVISE message has two purposes: either to provide execution and input/output overlap in order to reduce the number of page faults for a job, or to release pages of memory no longer required by that job. The ADVISE message is intended for use in improving job execution speed.

The ADVISE message indicates a virtual range (that is, a range of virtual addresses). The ADVISE message can be used in one of the following ways:

- As an ADVISE function to page in a virtual range.
- As an ADVISE function to page out a virtual range.
- As an ADVISE function to replace one virtual range with another.

Alpha(1)	r (16)	len (16)	unused (16)	0051 (16)
Alpha(2)	n (16)	eea (48)		
Alpha(3)	bl (16)	ba (48)		

r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1	Response code was issued by routine other than interrupt routine.
2	Illegal Beta address (only for n=4).
214	Beta buffer length error (only for n=4).

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion (two words). Used only if n=4.

n — Message options. Release the current interrupt and take the next interrupt in the stack, if one exists. When all interrupts outstanding have been processed, or if no other interrupts existed, do one of the following:

0	Return control to the interrupted program at the point of interruption.
1	Return control to the point following this particular RETURN FROM INTERRUPT message; that is, make the interrupt routine that issued this message the new level-0 routine.
2	Return control to the interrupted program at the address in register 4.
3	Abnormal termination control interrupt only. Abort at the point of original interrupt. Normal user dumps and trace-back information are produced for the original fatal error.
4	Return control to the interrupted program at the address in Beta(1) using data base information in Beta(2).

For n=4 only:

Beta(1)	unused (16)	vba (48)
Beta(2)	dbl (16)	dba (48)

vba — Virtual bit address in the interrupted program to which control is returned.

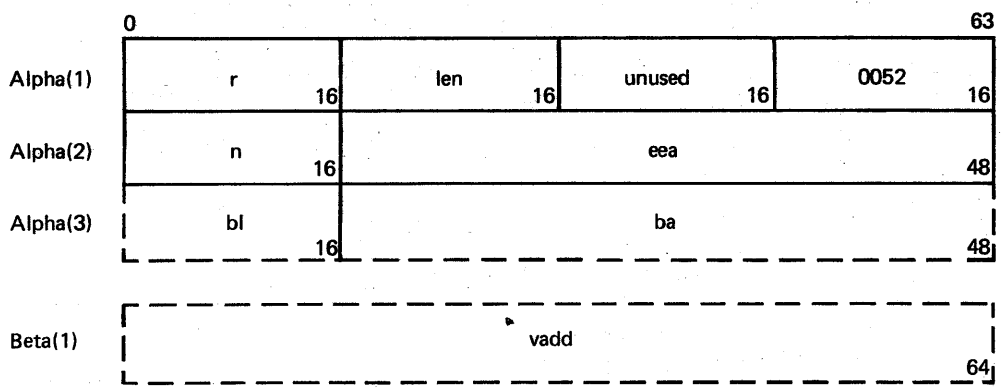
dbl — Length of data base to be reloaded.

dba — Address of data base to be reloaded for return to interrupted program.

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl,ba — If Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

Figure 4-46. RETURN FROM INTERRUPT Message Format



r — Response code; returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

- 1 — The vadd field does not contain the address of Alpha(1) of an EXPLICIT I/O message (f=0050)
- 214 — Beta buffer length error; either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If this field is #FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha(3), and len is the length in words of the Beta portion.

n — Message options:

- 0 — Give up the central processor until all input and output is complete; no Beta words are required for this option
- 1 — Give up the central processor until input/output calls specified by the Beta portion of the message are complete

eea — Virtual bit address to receive control if an error occurs during processing of this message (r≠0). If this field is zero when an error occurs, the error is considered fatal.

bl, ba — If the Beta and Alpha portions of the message are not contiguous (len=#FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

vadd — The virtual bit address of Alpha(1) of an EXPLICIT I/O message that must be completed before the central processor is temporarily given up.

Figure 4-47. GIVE UP CPU UNTIL I/O COMPLETES Message Format

The ADVISE in function is initiated without blocking the job from execution. Pages required to accommodate the ADVISE in request are obtained from the following categories in the specified order:

- Pages freed as the result of an accompanying ADVISE out function.
- Unallocated pages.
- Unmodified pages outside the working sets of connected tasks.
- Unmodified pages belonging to disconnected tasks.
- Modified pages belonging to the requestor but outside the requestor's working set.

If insufficient memory is available to accommodate the ADVISE in function, as many pages as possible are ADVISEd in and the user is informed that only a partial function was performed. The maximum number of pages that are read into memory by PAGER for any single ADVISE in function is 16 small pages or one large page. If the requestor exceeds the limit, the maximum number of pages ADVISEd in will be the limit. The user is returned the highest virtual small page address plus one in the specified virtual range that is in core after the ADVISE in function is complete.

The user should be mindful of the following points:

- Only a single Beta is allowed for an ADVISE in function in an ADVISE message.
- If a requested page is already in core, that page is ignored and the remaining pages are ADVISEd in.
- If a page fault occurs for an ADVISE in page, prior to its arrival in central memory, the system blocks the job from further execution until the page fault is satisfied.

- If a job has a machine-size working set, an ADVISE in function is accomplished by selecting the least recently used pages from within the job's working set as replacement pages.
- A virtual bit address that is not defined in any virtual map is considered to be a definition of new free space. An appropriate entry is made in the drop file map, and core space is allocated.

The ADVISE out function is used to remove a virtual range from memory and is initiated without the job being blocked from execution. All unlocked, modified pages within the specified virtual range are written to mass storage. The pages are then deleted from the page table. The user should be aware of the following restrictions:

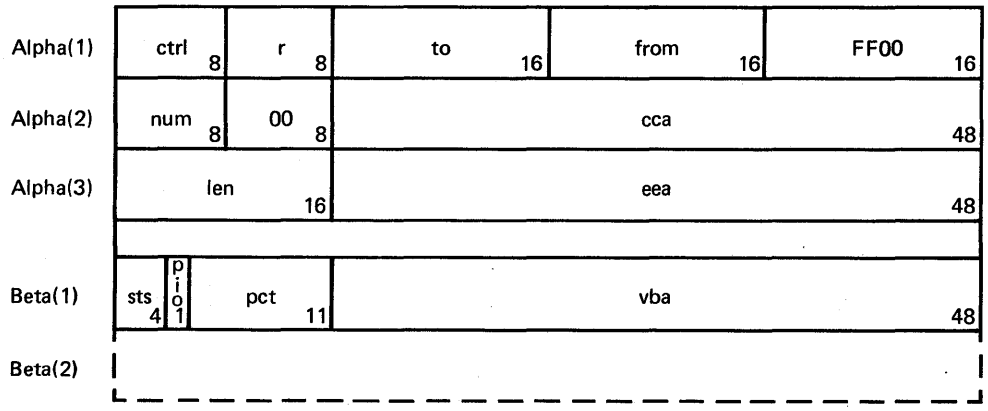
- Only a single Beta is allowed for an ADVISE out function in an ADVISE message.
- If a page within the specified virtual range is locked or not in memory, that page is ignored and the remainder of the request processed. If a locked page is detected, the user is informed.
- If a write access occurs for a page being written to disk as the result of an ADVISE out, the job is page blocked until the input/output is complete.

An ADVISE replace function is the combination of an ADVISE in function and an ADVISE out function in a single ADVISE message. The function can replace in full, or in part, one virtual range with another virtual range. The system first performs the ADVISE out function and then initiates the ADVISE in function. At a minimum, the number of pages freed by the ADVISE out function are then available for the ADVISE in function. If fewer pages are specified in the ADVISE out function than in the ADVISE in function, additional pages are selected by the system and paged out to provide sufficient pages to satisfy the ADVISE in function. The additional pages can be obtained from the following categories in the specified order:

- Unallocated pages.
- Unmodified pages belonging to connected tasks but outside their working sets.
- Unmodified pages belonging to disconnected tasks.
- Modified pages outside the requestor's working set.

It should be noted that two Betas are specified for the ADVISE replace function: one for the ADVISE out specification, and one for the ADVISE in specification, in any order.

The ADVISE message format shown in figure 4-48 is used when an ADVISE is issued directly to the system.



ctrl Should be set to zero. KERNEL will set this field to zero on return to the caller.

r Response code returned by the operating system after the specified ADVISE function has been processed. If no error is detected and the full request is processed, r is set to zero; otherwise:

1 Sts field in parameter set provides further status.
2 Illegal option specified for this message. Multiple ADVISE in or ADVISE out functions specified in a single message.
3 Address vba is out of the virtual address range for a user program.
4 Beta length error. Either the first word address of Beta plus length is out of the virtual address range for a user program or more than two parameter sets are specified.

to Should be set to zero. KERNEL will set this field to zero on return to the caller.

from Should be set to zero. KERNEL will set this field to zero on return to the caller.

Figure 4-48. ADVISE Message Format (Sheet 1 of 2)

num Number of Betas for this call:

- 1 A single ADVISE in or a single ADVISE out message
- 2 ADVISE replace message (that is, an ADVISE in message and an ADVISE out message with the order being immaterial)

cca Completion address for this call.

len Number of full words for each parameter set. Should be set to 1.

eea Virtual bit address to receive control if an error is detected during message processing (r≠0).

sts Status field, which is set to zero unless r is set to 1. In this case, the ADVISE function was incomplete and the code in this field indicates the cause:

- 1 ADVISE function is not complete due to more pages specified than a single ADVISE function can accommodate. The system ADVISEd 16 small pages or a single large page but did not satisfy the total request. ADVISE in Beta returned to the requestor contains the first small page address in the specified virtual range not processed. ADVISE out virtual range specification is unaltered. The status code 1 can be combined with any other status. For example, a status of 3 would indicate that more than 16 small pages or a single large page was specified in addition to a locked page within the specified virtual range being encountered while doing an ADVISE out function.
- 2 Partial ADVISE out function performed due to pages within the specified virtual range being locked.
- 4 Partial ADVISE in function performed due to insufficient memory resources to accommodate the request.
- 8 Page within specified ADVISE in range was found to already be in core.

pio Function requested:

- 0 ADVISE in function requested
- 1 ADVISE out function requested

pct Page count in small pages. PAGER determines what size page the requested virtual range is mapped into. If the request is for pages mapped into a large page, the entire large page is ADVISEd. If the request is for more than 16 small pages, PAGER will only ADVISE 16 pages and informs the requestor of this fact.

vba Virtual bit address of the start of the virtual range specified in the ADVISE function. vba is updated to the last page address plus one processed by an ADVISE in function (unaltered by an ADVISE out function).

Figure 4-48. ADVISE Message Format (Sheet 2 of 2)

PROCESS SYSTEM PARAMETER (f=#FF01)

The PROCESS SYSTEM PARAMETER system message can change or retrieve the value of one or more of the following system parameters in the file descriptor block (DB).

- C_RFL Current working set size limit
- C_MLP Maximum large page limit.
- C_RLP Current large page limit.

If the current large page limit is set less than the number of large pages currently assigned to the task, large pages are purged until the limit is reached.

The message format is shown in figure 4-49. The Alpha and Beta portions must be contiguous. Multiple Betas can be specified.

Alpha(1)	ctrl 8	r 8	to 16		from 16		FF01 16		
Alpha(2)	num 8	00 8	ca 48						
Alpha(3)	0001 16		ea 48						
Beta(1)	fc 16		opt 8	sts 8	val 32				

ctrl A value specified in this field is not used. KERNEL sets this field to zero on return to the caller.

r Response code returned by CYBER 200 OS when message processing is complete.

- 0 No errors.
- 1 An error occurred (refer to the sts field in the Beta(1)).
- 2 Illegal function code or option specified in Beta(1).
- 4 Beta buffer length error. Either the first word address of Beta plus its length is greater than the maximum virtual user address, or the Beta buffer is too small for the number of requests and length specified.

Figure 4-49. PROCESS SYSTEM PARAMETER (f=#FF01) (Sheet 1 of 2)

to	A value specified in this field is not used. KERNEL sets this field to zero on return to the caller.
from	A value specified in this field is not used. KERNEL sets this field to zero on return to the caller.
num	Number of parameter sets (Beta portions) for this call.
ca	Address at which execution continues following successful completion of the call. If zero, execution continues at the address following the call. If nonzero, execution continues at the specified address.
eea	Error exit address; virtual bit address to receive control if an error occurs during message processing (r≠0). If this field is zero when an error occurs, the task is aborted.
fc	Function code indicating the DB field value to be set or returned. 1 Maximum large page limit. 2 Current large page limit. 3 Current working set size limit.
opt	Message option. 0 Change the DB field value specified by fc to the value in val. If a controller exists for the task, its DB field value is also changed. 1 Return the DB value specified by fc in the val field.
sts	Beta status code. 0 No errors. 1 Requested current large page limit greater than maximum large page limit. 2 Requested current working set limit greater than maximum working set size limit. 3 Requested current working set limit too small to accommodate job's current large page limit. 4 Illegal operation requested.
val	New system parameter value if opt=0; field in which value is returned if opt=1.

Figure 4-49. PROCESS SYSTEM PARAMETER (f=#FF01) (Sheet 2 of 2)

The Virtual System Debug Tool (VSDT) provides a breakpoint capability to track the execution of the virtual portion of the CYBER 200 operating system. This facility is designed as a tool for the systems programmer who wishes to stop execution of the operating system at desired virtual system locations. This debugging aid provides the systems programmer with the capability to temporarily stop execution of the virtual system by setting read/write access breakpoints and execute breakpoints, and then to reset those breakpoints and continue execution.

The Virtual System Debug Tool is built as part of the virtual system of the CYBER 200 operating system. While the operating system is running, a debug command can be entered into a reserved shared table (T VSD) by using the Maintenance Control Unit (MCU) write command. This table is periodically checked by the resident portion of the operating system for commands. When a command is entered, control is passed to VSDT for handling. If an attempt is made to set a breakpoint at a paged-out address, that page will be paged-in by VSDT. All current breakpoint addresses can be found in the T VSD table.

RESIDENT SYSTEM

The resident system periodically checks the contents of the VSDT input buffer word of the T VSD table for a nonzero value. Upon detection of a nonzero value, control is passed to VSDT which examines the input buffer word. If it contains "COMMAND", the debug command buffer is checked for a debug command. Any existing debug command is then processed by VSDT.

If the input buffer word contains "CONTINUE" (normally entered after a breakpoint has been hit and appropriate sections of memory have been displayed), VSDT resumes execution of the virtual system. The status word of the T VSD table (containing "STOP") will be cleared. If "CONTINUE" is entered without a breakpoint having been hit, the error message word in the T_VSD table will contain the message "ILL COMM" (Illegal Command) and control will return to the resident system.

VIRTUAL SYSTEM

VSDT handles the setting and resetting of breakpoints, as well as the resumption of virtual system execution. When control is passed from the resident system to VSDT, VSDT checks the input buffer word in the T_VSD table for either "COMMAND", in which case it will process debug commands, or, "CONTINUE", for resumption of virtual system execution after a breakpoint has been hit.

If a breakpoint command has been entered, VSDT will put the breakpoint address into the T_VSD table; save the instruction at the breakpoint address; set up the breakpoint jump instruction; and clear the word "COMMAND" from the input buffer word. If a command to reset the breakpoint has been entered, VSDT will restore the original instruction at the breakpoint address, as well as clear the breakpoint address in the T_VSD table.

USER AND SYSTEM INTERFACES

The Maintenance Control Unit (MCU) is used as the input device for the Virtual System Debug Tool. Debug commands are entered in ASCII format into the T_VSD table by the MCU A command. To stop the CYBER 200 operating system after system autoload, the user can enter a breakpoint command into the T_VSD debug command buffer. When the execution of the virtual system is stopped, MCU memory commands are used for system examination.

This discussion assumes that the user is familiar with the Maintenance Control Unit (MCU) and its commands. Refer to the Maintenance Control Unit reference manual for additional information.

SHARED TABLE

An additional table (T_VSD) has been added into the shared table list. The main function of this table is communication between the user and the resident system. The structure of the T_VSD table is given in table 5-1.

COMMANDS

Since all debug breakpoint commands are entered by MCU memory alteration commands, systems programmers are responsible for entering debug commands into the appropriate locations in the T_VSD table.

The commands supported by VSDT are shown in table 5-2.

COMMAND FORMAT

The following conventions must be observed when entering the breakpoint commands:

A blank must be used to separate the command and its first parameter.

Brackets [ ] are used to indicate an optional parameter. Defaults are assumed for omitted parameters and are defined in the command descriptions. Either no parameters or any single parameter can be selected.

TABLE 5-1. STRUCTURE OF THE T_VSD TABLE

Word	Name	Word	Name
1	W_VSDNAME	11	W_VSDC6
2	W_VSDINPUT	12	W_VSDC7
3	W_VSDBSTAT	13	W_VSDAB1
4	W_VSDBADD	14	W_VSDAB2
5	W_VSDERRMES	15	W_VSDAB3
6	W_VSDC1	16	W_VSDAB4
7	W_VSDC2	17	W_VSDAB5
8	W_VSDC3	18	W_VSDAB6
9	W_VSDC4	19	W_VSDABA
10	W_VSDC5	20	W_VSDREG

Variable	Word	Description
W_VSDNAME	1	Contains the name "T_VSD" (left-justified with blank fill) identifying the table.
W_VSDINPUT	2	System command input buffer - cleared to zeros after command is processed. "COMMAND" - Process VSDT debug command which has been entered in the T_VSD debug command input buffer. The word "COMMAND" is left-justified with blank fill. "CONTINUE" - Continue execution of the virtual system after a breakpoint has been hit. The word "CONTINUE" is left-justified with blank fill.
W_VSDBSTAT	3	Virtual System status word - contains the word "STOP*A" or "STOP*X" (left-justified with blank fill) when breakpoint is hit. "STOP*A" indicates that an access breakpoint had occurred while "STOP*X" indicates that an execute breakpoint had occurred. The status word is cleared to zeros upon continuation of Virtual System execution.
W_VSDBADD	4	Contains the last breakpoint address hit by Virtual System (right-justified with zero fill). For an execute breakpoint, it will be the address of the instruction causing the breakpoint. For an access breakpoint, it will be the address of the last executed instruction which can be up to 35 instructions past the instruction causing the breakpoint.
W_VSDERRMES	5	Error messages returned by VSDT, left-justified with blank fill. Cleared to zeros when next command is entered.
W_VSDC1	6	Start of debug command buffer - commands are entered in ASCII mode - left-justified with blank fill. Cleared to zeros after a command is processed. If an error has occurred, the buffer is not cleared to zeros. The command can then be reentered in its entirety, or edited in place. In either event, if any extraneous characters remain after the corrected command has been placed in the command buffer, they must be blanked out by use of the MCU A command.
W_VSDC2	7	Continuation of debug command buffer.
W_VSDC3	8	Continuation of debug command buffer.
W_VSDC4	9	Continuation of debug command buffer.
W_VSDC5	10	Continuation of debug command buffer.

TABLE 5-1. STRUCTURE OF THE T_VSD TABLE (Contd)

Variable	Word	Description
W_VSDC6	11	Continuation of debug command buffer.
W_VSDC7	12	End of debug command buffer.
W_VSDAB1	13	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDAB2	14	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDAB3	15	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDAB4	16	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDAB5	17	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDAB6	18	Contains address of breakpoint which has been set in the Virtual System (right-justified and zero-filled).
W_VSDABA	19	Address of the read/write access breakpoint (right-justified and zero-filled).
W_VSDREG	20	Address of the Virtual System Register File (right-justified and zero-filled).

Notes: 1. The upper 16 bits of W VSDBADD contain a value equal to the number of times the breakpoint has been hit. When the value reaches the count specified by the 'n' option, the system will halt and the appropriate message will be placed in W_VSDBSTAT.

2. The upper 16 bits of W VSDAB1-6 and W VSDABA contain the value specified by the 'n' option, which is the number of times a breakpoint is to be executed before the system is stopped.

TABLE 5-2. VSDT COMMAND SUMMARY

Command	Explanation
BKPT	Set execute breakpoint.
BKR	Set read access breakpoint.
BKRW (BKWR)	Set read and write access breakpoint.
BKW	Set write access breakpoint.
COMMAND	Process virtual system debug commands.
CONTINUE	Continue execution from last user breakpoint.
RBKP	Remove all or selected breakpoints.
PAGEIN	Bring page containing given virtual address into memory.

Note: Underscored letters indicate short form of the command.

Underscored letters of each command or keyword parameter indicate the minimum character string used to specify the command or parameter. Any number of characters, from the minimum string to the entire word, can be used. For example, BK, BKP, and BKPT will all result in execution of the BKPT command.

All address, count, and number parameters are specified in hexadecimal.

Commands are entered into T_VSD starting with the leftmost byte in the command buffer.

Blanks are not used between subsequent parameters (with the exception of the blank used to separate the command and its first parameter). Commas are used to separate all subsequent parameters.

The virtual address parameter for debug commands is assumed to have an offset bit address C00000000000 in hexadecimal; that is, C00000000000 will be added to the value entered by the systems programmer to derive the virtual system address.

DEBUG COMMANDS

Debug commands are entered into W_VSDC1 through W_VSDC7.

Up to six execute breakpoints are allowed in the virtual system at any time. These are software breakpoints and cause the virtual system to stop execution prior to executing the instruction at the specified virtual address.

Only one access (read, write, or read/write) breakpoint can be set at any one time. The access breakpoint will use the hardware breakpoint instruction (op code #04), and will cause the virtual system to stop executing whenever the super-word at the specified virtual address is accessed. If a "value" has been specified, the virtual system will stop only if the data is equal to the value, after the mask has been applied to the data.

If an attempt is made to set more than one access breakpoint, only the most recent breakpoint specified will be set. All virtual addresses specified for an access breakpoint will be truncated to a super-word boundary.

The user can specify the number of times a breakpoint is reached before execution of the virtual system is stopped.

The format of the command to be entered at the MCU is:

△ T_VSD entry address △ VSDT command △ parameters

The command buffer and input word are cleared upon acceptance of the command.

VSDT commands and associated parameters are shown in table 5-3.

Breakpoints can be removed through use of other specific commands shown in table 5-4. The breakpoints can be removed individually or all together.

CONTROL COMMANDS

Control commands are entered into W_VSDINPUT which allow the user to input debug commands into VSDT, and continue execution after a breakpoint has been reached.

Table 5-5 shows the command for accessing paged-out addresses.

Debug commands are processed by VSDT by using the MCU command A to write the word "COMMAND" into W_VSDINPUT following entry of the debug command into the command buffer. The debug commands are entered initially into the T_VSD table or after a breakpoint has been hit.

TABLE 5-4. VSDT COMMAND FOR RESETTING BREAKPOINTS

VSDT Command	Parameters	Significance
RBKP	ALL RW X virtual address	RBKP allows the user to remove all breakpoints that have been set (ALL), to remove all execute breakpoints that have been set (X), to remove any read or write breakpoints that have been set (RW), or to remove a single breakpoint set by virtual address. Default is ALL.

TABLE 5-3. VSDT COMMANDS FOR SETTING BREAKPOINTS

VSDT Command	Parameters	Significance
BKPT	virtual address (,n)	BKPT sets an execute breakpoint which causes the virtual system to stop execution at the specified address. Up to six breakpoints can be set at one time. Default is n=1.
BKR	virtual address (,n) or virtual address (,=value(,mask)) or virtual address (,n(,=value(,mask)))	BKR sets a read breakpoint. Execution of the virtual system will stop when the specified super-word is read from. Default is n=1.
BKW	virtual address (,n) or virtual address (,=value(,mask)) or virtual address (,n(,=value(,mask)))	BKW sets a write breakpoint. Execution of the virtual system will stop when the specified super-word is written into. Default is n=1.
BKRW BKWR	virtual address (,n) or virtual address (,=value(,mask)) or virtual address (,n(,=value(,mask)))	Either BKRW or BKWR sets the read/write breakpoint. Execution of the virtual system will stop when the specified super-word is read from or written into. Default is n=1.

TABLE 5-5. VSDT COMMAND FOR ACCESSING PAGED-OUT ADDRESSES

VSDT Command	Parameters	Significance
PAGEIN	virtual address	PAGEIN causes the virtual system to access the specified virtual address causing, in some cases, a page fault. This command is used to bring a page containing the specified virtual address into memory.

The CONTINUE command allows execution to continue after a breakpoint has been reached. Execution can be continued by using the MCU A command to write the word "CONTINUE" into W_VSDINPUT in the T_VSD table.

The user can also obtain breakpoint status information. No specific command is needed. Since breakpoint addresses are stored in the T_VSD table, the MCU "DISP addr" command can be used to check for breakpoints set.

ERROR MESSAGES

Error messages which are placed in W_VSDERRMES of the T_VSD table are shown in table 5-6. The user can input the proper command in its entirety after diagnosing the error message or edit it in place. W_VSDERRMES is cleared after the acceptance of subsequent commands.

TABLE 5-6. VSDT ERROR MESSAGES

Error Message	Significance
ILL COMM	Illegal command.
ILL PARM	Illegal parameter.
ILL NUM	Illegal number.
TAB FULL	Breakpoint table is full.
ILL ADDR	Illegal address specified in RBKP command.
NO ADDR	No address has been specified in a breakpoint instruction.

ANALYZER is a tool for extracting information from a system dump. It generates an output file containing the following information, in the order listed:

- A hexadecimal dump of the system tables and dynamic stack, with ASCII interpretation.
- A hexadecimal dump of any core areas specified with the CORE parameter in the ANALYZER execute line.
- A dump of the page table; four listings of the page table entries are provided, sorted by:

  Page table bit address

  Physical page number

  Virtual address and key

  Key and virtual address

- A dump of the history table with entries listed in chronological order; time is reported at installation option.
- A dump of the resident operating system dynamic stack.
- A dump of the virtual system minus page, register file, and dynamic stack.
- A dump of the user's minus page, second minus page, register file, and dynamic stack, for any active user.

Input for ANALYZER is typically constructed by the following steps:

1. A file is created the size of core, and a utility (such as the FILES utility described in volume 1 of this manual) is used to determine the unit number and absolute disk address of the file.
2. With the system stopped, a station command (such as the WCEN command) is used to copy core to the precreated file.

ANALYZER can be used to examine the file after the system has been recovered or reautoloaded.

The format of the ANALYZER execute line is shown in figure 6-1. All parameters are optional and order-independent.

ANALYZER(I=ifn,L=ofn,CORE=1_1,h_1, . . . ,1_4,h_4)

ifn	Name of the input file containing a dump of core. The default is I=DUMPFILE.
ofn	Name of the file to contain ANALYZER output. The default is L=OUTPUT. Any existing file with the same name as the output file is destroyed and a new file with that name is created.
1_i,h_i	Additional area of core to be dumped. As many as four 1_i,h_i pairs can be specified. Words 1 through h, inclusive, are dumped as follows:
	1 Low hexadecimal bit address of the area to be dumped
	h High hexadecimal bit address of the area to be dumped

Figure 6-1. ANALYZER Execute Line Format

The accounting system is that part of the virtual system that gathers system resource usage statistics and records them at two locations: in a cumulative accounting buffer, and in an active accounting file. The statistics recorded in the cumulative accounting buffer are a subset of those recorded in the active accounting file. Statistics recorded in the cumulative accounting buffer can be retrieved by a user or utility program only when the job for which the statistics are being gathered is running; statistics recorded in the active accounting file can be retrieved by an installation-defined user at any time after the file is no longer active.

CYBER 200 OS provides two units with which to measure resource usage, system time units (STUs) and system billing units (SBUs). As described later in this section, the SBU calculation uses a variable rate accounting factor and a service level index while the STU calculation does not.

The information recorded in the accounting file is designed to be accessed by an installation-defined accounting program. No CYBER 200 OS utilities currently exist to use these statistics.

CALCULATION OF STUs USED

A system routine called TCHARGE calculates the number of STUs consumed by a task. To calculate the STUs consumed, it first multiplies each system resource usage quantity by the weighting factor for that resource and then adds the products together.

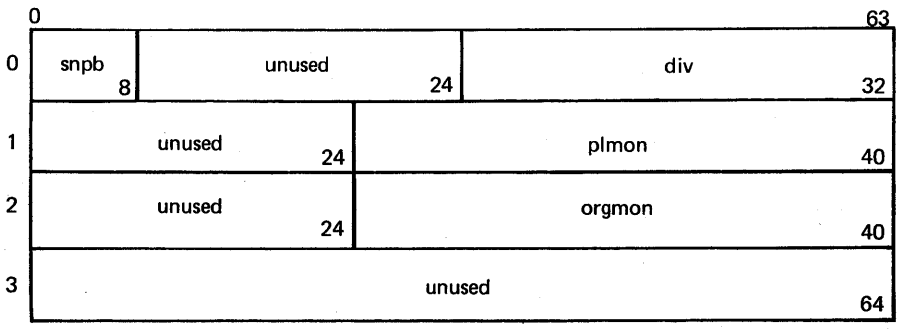
The system resources and their units of measure are listed in table 7-4. The weighting factors are specified by installation parameters. A weighting factor of zero eliminates the corresponding system resource from the STU calculation.

A system second is one million STUs.

REPOSITORIES

Each operating system repository table entry is loosely referred to as a repository and specifies a quantity of system time units (STUs) allocated to a division of users or an individual user by the installation administrator. The default units for STUs are microseconds. The format of a repository is shown in figure 7-1.

The accounting system also monitors the repositories of all users currently logged onto the operating system. A user number must jointly or individually be associated by an installation administrator with a repository before the operating system can be accessed under that user number. No utilities are currently provided as part of the operating system to use the resource usage statistics for decrementing the repositories of active users, although certain interactive capabilities are available for replenishing a repository and finding out what the current value in the repository is.



snpb	Snap-back flag. At user logoff, if this field is zero, return money to the user's private account; and if it is nonzero, or if the user directory indicates it, return money to the repository.
div	Division name, in ASCII.
plmon	Quantity of STUs currently available to the division.
orgmon	Original quantity of STUs given to the division.

Figure 7-1. Repository Table Entry Format

An installation administrator has the responsibility for initializing repositories and making the association of user numbers with repositories. The association of a repository with a user number is specified in the user directory entry for the user number. Other information that is retained in the user directory and that might be of use to an accounting utility are the user number's current private STU limit, the percentage draw allowed against the repository, the number of STUs used by this user for this logon, and a flag indicating where unused STUs are to be returned when this user logs off. Information in the user directory is retrieved and maintained through the UPDATE USER DIRECTORY message (f=0023) and EXECUTE OPERATOR COMMAND message (f=0021).

A batch user can replenish a repository by issuing option C of the MISCELLANEOUS message. Also, before a user can access the operating system with his or her own user number, the user (under another user number, which must be privileged), an installation-defined utility, or the operator must withdraw an allocation of system resource charge units from the appropriate repository. This draw can be performed by a batch user via the MISCELLANEOUS (f=0024), UPDATE USER DIRECTORY (f=0023), or EXECUTE OPERATOR COMMAND (f=0021) message. An interactive user can issue an (sc)G+xx command from a terminal to withdraw a quantity from the repository. (See volume 1 of this manual.)

A user can withdraw from the repository a maximum of (p/100)*orgmon, where p is the value (recorded in the user directory) of the maximum percentage draw from the repository allowed for that user. The amount withdrawn must be less than the difference between (p/100)*orgmon and the amount already used by this user. After the initial draw, the user can withdraw a maximum of (p/100)*plmon.

A batch user can find out how many STUs are specified by the repository by issuing option 1 of the MISCELLANEOUS message. An interactive user can get the same information by issuing an (sc)BB command from a terminal. (See volume 1 of this manual.)

Any job running when a draw is used up will abort immediately.

STATISTICS ACCUMULATION

Having gained access to the operating system, a user can then execute tasks and jobs. Statistics are gathered on an event basis and are accumulated over each task or batch job in both the cumulative accounting buffer and the active accounting file. An installation administrator has the option of using the statistics in the buffer to debit the STUs withdrawn by the user from the repository, or of using the equivalent statistics in the active accounting file to charge the user. The buffer statistics are also provided as information to the user via the USER/ACCOUNTING COMMUNICATION (f=000E) and MISCELLANEOUS (f=0024) messages.

CUMULATIVE ACCOUNTING BUFFER

The cumulative accounting buffer is provided so that resource usage statistics for a task or job are available during the time that the task or job is actually running. The buffer contains one 16-word entry for each active task (each descriptor block). The fields in each entry are updated as the tasks use up various system resources. When a job has finished, the entries for that job no longer exist in the cumulative accounting buffer; the entries exist only for the duration of the job.

A user is given access to the statistics in the cumulative accounting buffer for his or her running tasks alone via the USER/ACCOUNTING COMMUNICATION message. The virtual system returns the current accumulation of all statistics in the buffer to a user that issues this message with the c field set to 3. Based on these statistics, it might be possible to modify user programs to reduce system resource usage and perhaps reduce charges. This reduction would be dependent on the installation-defined charge algorithm.

In addition to individual task statistics, total job statistics for all controllee tasks of a batch job are available from the cumulative accounting buffer. These cumulative job totals are stored in the batch processor's entry in the buffer. By issuing the USER/ACCOUNTING COMMUNICATION message with c=3, the batch processor can access the total raw statistics for the duration of a batch job. These unfactored values can be used as input to an installation-provided routine that computes the charges for the job. Any of these charges, raw statistics, or factored charges might then be printed on the job's dayfile.

ACCOUNTING FILE

The accounting file is a history file of operating system resource usage, including tape and disk file usage, task and job execution times, terminal usage, and autoload and recovery events. Records are written into the file as resources are used. Accounting record types are listed in table 7-1.

More than one accounting file can exist; all such files are unattached permanent files belonging to the operator; however, only one file, named *AF, can be active at any one time. If during system initialization no active accounting file *AF exists, one is created at that time. A second file, *AF2, is also created. When the current active accounting file becomes full, it is renamed, file *AF2 becomes the active accounting file *AF, and a new *AF2 is created. The size of all accounting files is determined by the system parameter AFSIZE.

The accounting file can be used as follows:

- For each task, the accounting system writes a begin task accounting record to the accounting file. A unique task number is assigned to each task and is placed in all accounting records associated with the task. Each accounting period, the system writes a new task field overflow record for the task or else updates an existing one. When the task completes, the system writes an end task accounting record.
- To accumulate raw statistics for the duration of a task, the installation's program must locate on the accounting file all task field overflow records for a unique task number, extract the desired statistics, and sum them for each overflow record for the task.

- At the start of each batch job, the batch processor issues an option of the USER/ACCOUNTING COMMUNICATION message (f=000E), and a job start accounting record is written to the accounting file. The header of this record contains the user number and suffix under which the batch processor is running. Any controllee of this batch processor runs under this user number and suffix. The task record headers for these controllee tasks contain these identifiers also. At the end of a batch job, the batch processor issues another option of the USER/ACCOUNTING COMMUNICATION message, and a job end accounting record is written to the accounting file.

To accumulate raw statistics for the duration of a batch job, the installation's accounting program must locate the job start and job end records on the accounting file, and record the user and suffix numbers. Any task overflow records (between the job start and job end records) containing this user and suffix number are for tasks of this batch job. Statistics can be extracted from each of these overflow records and summed to get the cumulative batch job statistics. These totals are equivalent to those extracted from the batch processor's entry in the cumulative accounting buffer.

TABLE 7-1. ACCOUNTING RECORD TYPE AND SUBTYPE CODES

Subtype	Record Type Code					
	Task 1	Terminal 2	Disk File 3	Magnetic Tape 4	System 5	Job 6
0	Begin task	Logon	Open	Assignment	Autoload	Job start
1	End task	Logoff	Close	Release	Recovery	Job end
2	Field overflow		Create		New date entered	
3		notmsg/ notwds field overflow	Destroy		New time entered	
4			Extend			
5			Reduce			
6			Change user number			
7			Change file name			
8			Change account number			

Active Accounting File Blocks

Accounting records are accumulated in a 512-word memory buffer (this buffer is not the cumulative accounting buffer). The buffer is written to the accounting file when the buffer is full, or when an entry is made in the buffer after more than 5 minutes have elapsed since the previous buffer was written to the accounting file. The format of the buffer and, therefore, of the accounting file record blocks, is shown in table 7-2.

The master clock entries in words 0 and 3 of an active accounting file block are eight decimal numbers indicating the year, month, day, hour, minute, second, millisecond, and an installation-defined machine designator. The entries are represented as 16 digits in hexadecimal form within one word, as shown in figure 7-2. For example, the date November 6, 1975, the time 9:10:20.623, and the machine designator 1 would appear as shown in figure 7-3.

Table 7-3 lists the information kept in the first block of *AF, including the name to which the name *AF is to be changed when the file becomes full, and the address of the block in which space is currently available. Accounting records are written sequentially into *AF beginning with the second block. The record type and length are indicated by the first word of each record, as shown in figure 7-4.

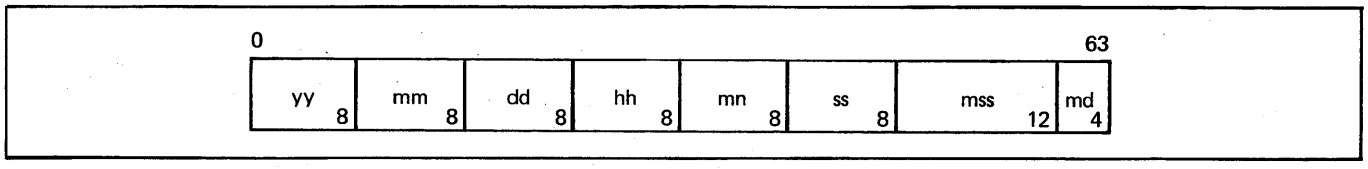


Figure 7-2. Master Clock Format

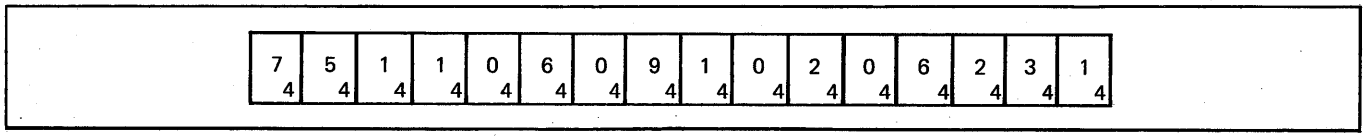


Figure 7-3. Master Clock Example

TABLE 7-2.
ACTIVE ACCOUNTING FILE BLOCK FORMAT

Word	Contents
0	Master clock time and date information at the time that the first entry is written to the buffer.
1	Microsecond central processor clock reading at the time that the first entry is written to the buffer.
2	Reserved for system use.
3	Master clock time and date information at the time that the last entry is written to the buffer.
4	Microsecond central processor clock reading at the time that the last entry is written to the buffer.
5	Cumulative system CPU overhead in microseconds. The sum of KERNEL, PAGER, and virtual system CPU times.
6	Cumulative USER CPU time in microseconds.
7	Cumulative WAIT time in microseconds. The CPU is available for user execution but all user tables are waiting for I/O completion.
8	Cumulative IDLE time in microseconds. The CPU is available for user execution but no user tasks are waiting for the CPU.
9 to #1FE	Filled sequentially with accounting records. Unused words contain binary zero.
#1FF	This word is set to contain all hexadecimal 2's when the buffer is full.

Task Records

Task accounting (type 1) records are written in the accounting file whenever a task is started or ended, or when task accounting information is available and no task field overflow record exists in the current accounting file block. A task record consists of a four-word header plus additional words, depending on the record subtype. The header format is shown in figure 7-5.

When the subtype code is 0 (begin task), the record words are as shown in figure 7-6. When the subtype code is 1 (end task), only the record header is used. All task accounting information is accumulated in task field overflow records before an end task record is written.

A task field overflow (subtype 2) record contains raw accounting statistics accumulated for the duration of a task. When the operating system calls the accounting system with task accounting information, the accounting system searches for a task field overflow record pertaining to the specified task. If such a record is found, its fields are updated with the new information. If no task field overflow record exists for the specified task, such a record is created and filled with the current information. The record words are shown in figure 7-7.

Each task field overflow record field (except for the ferr field) is cumulative over the accounting periods for the task until potential overflow is detected. For cumulative values for an entire task, the appropriate field should be summed over all of the task's task field overflow records in the accounting file. The values for the ferr and wssiz fields for the entire task are in the last record for the task.

The number of disk sectors transferred for large page implicit reads can be readily computed from the value of the lpflt field.

Terminal Records

Terminal accounting (type 2) records are entered into the accounting file whenever a user logs on or off a terminal, or when the notmsg or notwds fields in the user table are about to overflow. A terminal record consists of a four-word header plus one additional word. The format of the header is the same as for the task record header, except that the type field is 2, the subtype field can be 0 (logon), 1 (logoff), or 3 (notmsg or notwds field overflow), and the controllee level, suffix, job type, and task number fields are unused.

When the subtype code in the record header is 0 (logon), the fourth word in the record is as shown in figure 7-8.

When the subtype codes are 1 (logoff) and 3 (notmsg/notwds field overflow), the fourth word in the record is as shown in figure 7-9.

Disk File Management Records

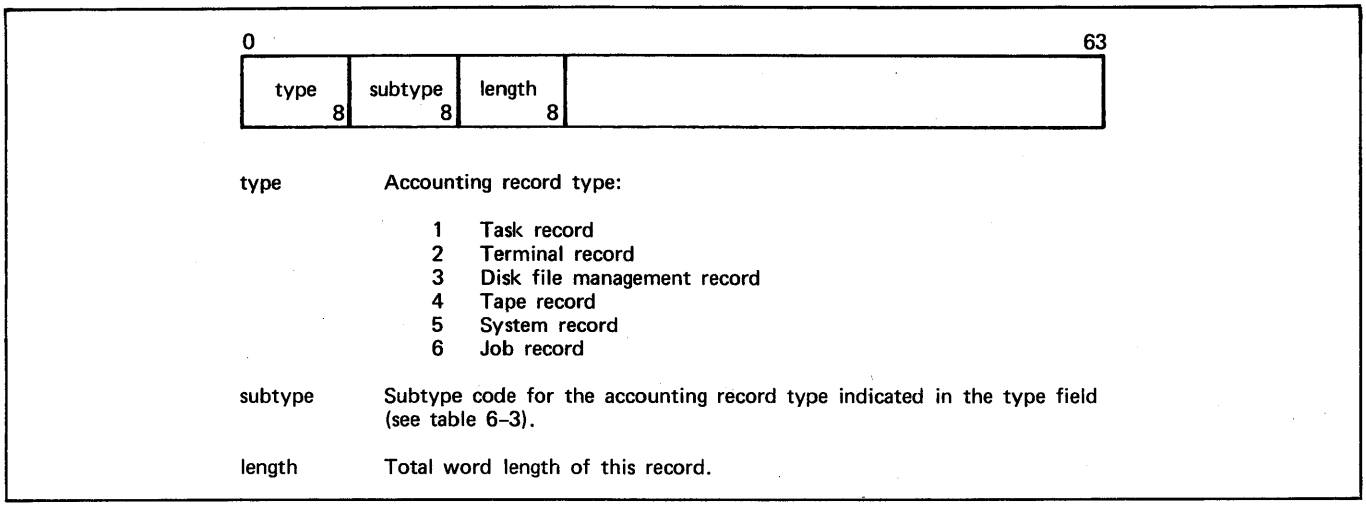
Disk file accounting (type 3) records are issued whenever there is non-data-transfer activity for a file. Disk file records consist of a four-word header plus additional words, depending on the record subtype. The format of the header is the same as for the task record header (figure 7-5), except that the type field is 3, and the subtype field can be 0 (open), 1 (close), 2 (create), 3 (destroy), 4 (extend), 5 (reduce), 6 (change user number), 7 (change file name), or 8 (change account number).

The format of the additional words (words 4 and 5) when the subtype codes are 0 (open), 1 (close), and 2 (create) is the same as the format of a task record of subtype 0 (figure 7-6), except that the file named can be a data file as well as a code (executable) file.

Three additional words are required (words 4, 5, and 6) when the subtype codes are 3 (destroy), 4 (extend), and 5 (reduce). The first two words are the same as words 4 and 5 for subtypes 0, 1, and 2. The third word consists of two 32-bit fields. The leftmost field gives the number of 512-word blocks in the file at the time that the destroy, extend, or reduce record is written. The rightmost field gives the number of seconds that this file existed under the user number, file name, account number, and size specified.

TABLE 7-3. FORMAT OF FIRST BLOCK OF ACTIVE ACCOUNTING FILE

Word	Contents	Word	Contents
0	Ayydddnn, the name to be given to this file when it is deactivated. yyddd is the year and day number computed when the file is activated; nn is a sequence number, modulo 100.	4	Reserved for system use.
		5	Reserved for system use.
		6	Microsecond central processor clock value when this file is deactivated.
1	Name of the last accounting file, in the form Ayydddnn. At autoload, this field contains binary zero if no *AF file exists at that time.	7	ASCII date when this file is deactivated, in the form mm/dd/yy (month/day/year).
		8	ASCII time when this file is deactivated, in the form hh:mm:ss (hours:minutes:seconds).
2	Name of the next accounting file, Ayydddnn.		
3	Relative block address of the most recently created block in the file, where the header block is block 0.	9 to 1FF	Unused.



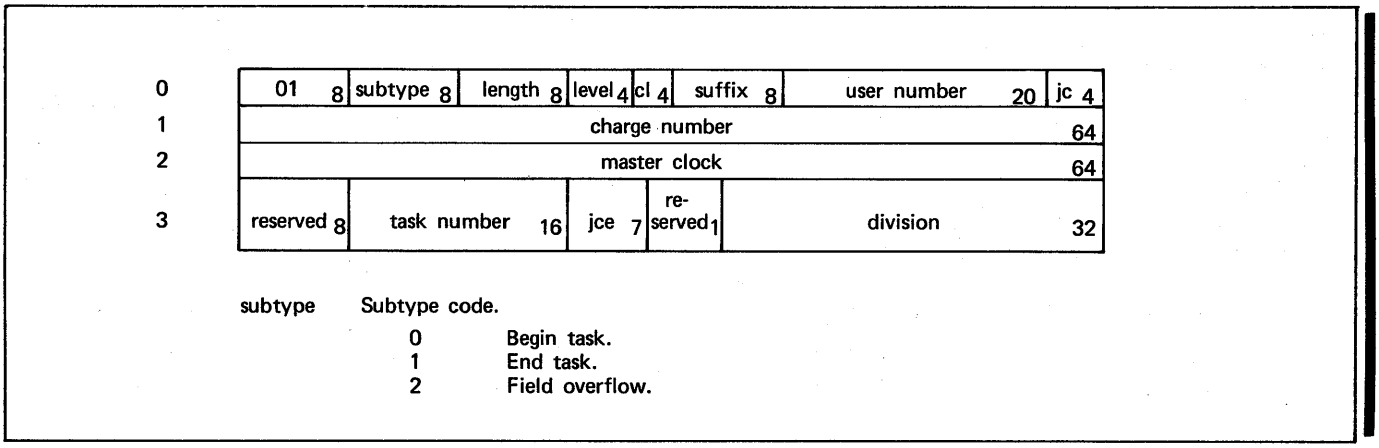
type — Accounting record type:

1 Task record
2 Terminal record
3 Disk file management record
4 Tape record
5 System record
6 Job record

subtype — Subtype code for the accounting record type indicated in the type field (see table 6-3).

length — Total word length of this record.

Figure 7-4. General Format of Accounting Record First Word



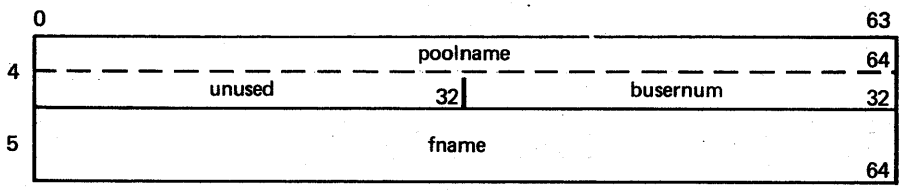
subtype — Subtype code.

0 Begin task.
1 End task.
2 Field overflow.

Figure 7-5. Task Record Header Format (Sheet 1 of 2)

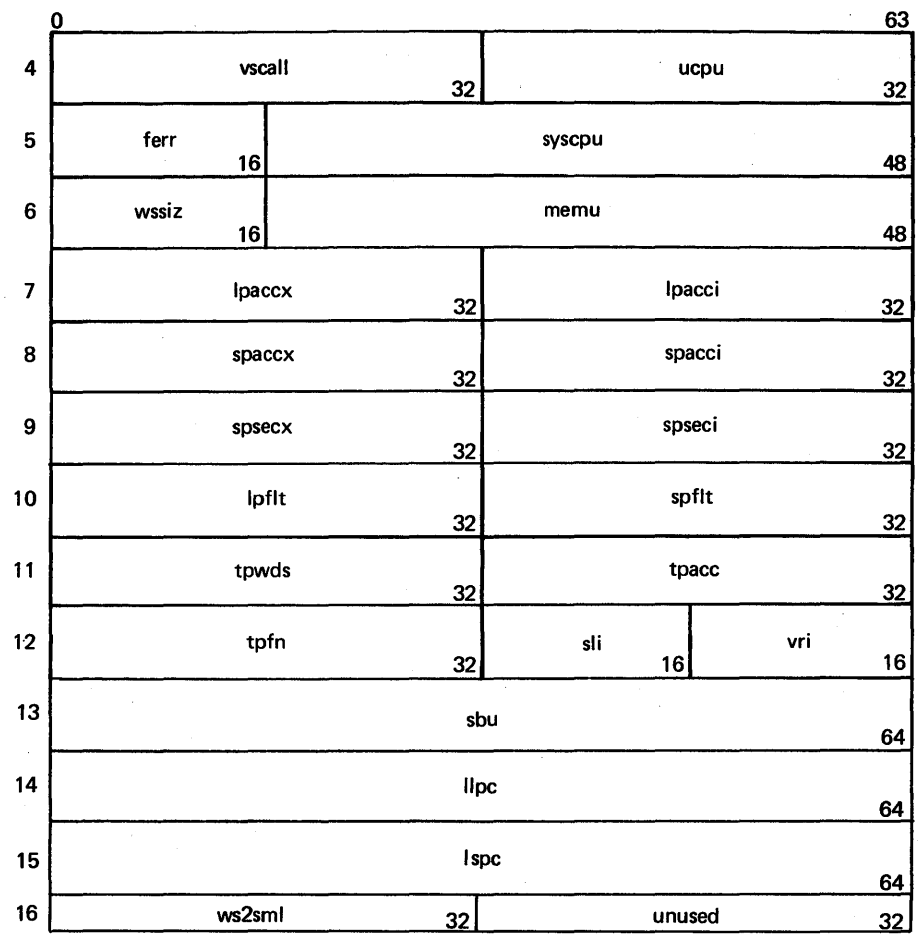
length	Total word length of this record.
level	Job level (1 through 15).
cl	Controllee level (1 through 9).
suffix	Suffix to which the task is attached. 1 Suffix A. 2 Suffix B. 3 Suffix C. 4 Suffix D.
user number	Binary user number for this task.
jc	Job class. 1 Standby. 2 Batch. 3 Interactive. 4 High priority. 5 System.
charge-number	Account identifier from logon line, STORE card, or link station job statement; identifies account to which the task is being charged.
master clock	Value of the master clock when this record was created.
task	Number assigned when the begin task record is written. All subsequent accounting records pertaining to this task contain this number. (The accounting system assigns task numbers sequentially.)
jce	Job category entry number (0 through 65).
division	Division code (in ASCII) under which this task is running. (A user's division is established when his user directory is created.)

Figure 7-5. Task Record Header Format (Sheet 2 of 2)



poolname busernum	Name of the pool in ASCII, if this file is a pool file member; or binary user number of the task file owner, if this is a private or public file.
fname	Name of the task file in ASCII, left-justified with blank fill.

Figure 7-6. Task Record Format, Subtype 0



vscall — Number of virtual system user calls made.

ucpu — User execution CPU time, in microseconds.

ferr — Error number of the fatal error condition, transferred from word 139 of the user's minus page. This field is zero if there is no error.

syscpu — Virtual CPU time, in microseconds, for user program execution.

wssiz — Average working set size, in small pages.

memu — Memory usage (at the end of each accounting period, (current working set size) *(user CPU time for current accounting period) is computed.)

lpacci — Number of disk accesses (output requests issued) for large page implicit writes.

spaccx — Number of disk accesses (input/output requests issued) for small page explicit reads and writes.

spacci — Number of disk accesses (output requests issued) for small page implicit writes.

spsecx — Number of disk sectors transferred for explicit reads and writes.

spseci — Number of disk sectors transferred for implicit writes.

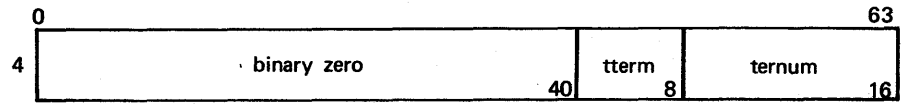
lpflt — Number of disk accesses (input requests issued) that resulted from large page faults (large page implicit reads).

spflt — Number of disk accesses (input requests issued) that resulted from small page faults (small page implicit reads).

Figure 7-7. Task Record Format, Subtype 2 (Sheet 1 of 2)

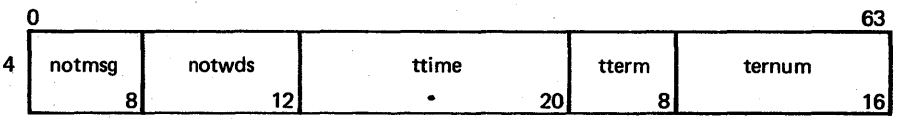
tpwds	Number of 16-bit bytes transferred to or from tape files.
tpacc	Number of tape accesses (input/output requests issued) for reads and writes.
tpfn	Number of nonread and nonwrite tape functions, such as read hardware status.
sli	Service level multiplying index used for variable rate accounting.
vri	Variable rate index used for variable rate accounting.
sbu	Floating point summation of tasks system billing units (summation of MVAL's returned from MCHARGE).
llpc	Number of large pages lost to other tasks.
lspc	Number of small pages lost to other tasks.
ws2sml	CPU time in microseconds that the task's working set size appeared to be too small.

Figure 7-7. Task Record Format, Subtype 2 (Sheet 2 of 2)



tterm	Reserved for operating system use.
ternum	Reserved for operating system use.

Figure 7-8. Terminal Record Format, Subtype 0



notmsg	Number of messages to or from the terminal since logon or since a previous overflow record.
notwds	Number of words transferred to or from the terminal since logon or since a previous overflow record.
ttime	Time the terminal has been in use, in seconds, since logon or since a previous overflow record.
tterm	Reserved for operating system use.
ternum	Reserved for operating system use.

Figure 7-9. Terminal Record Format, Subtypes 1 and 3

The format of the additional words when the subtype codes are 6 (change user number), 7 (change file name), and 8 (change file account number), is the same as words 4, 5, and 6 for subtypes 3, 4, and 5. Word 7 contains the new binary user number for subtype 6, the new file name for subtype 7, and the new account number for subtype 8.

Tape Records

Tape accounting (type 4) records are entered into the accounting file whenever a tape is assigned to, or released from, a task. Tape records consist of a four-word header, plus additional words depending on the record subtype. The format of the header is the same as for the task record header (figure 7-5), except that the type field is 4, and the subtype field can be 0 (assignment), or 1 (release).

When the subtype code is 0 (assignment records), only the record header is used. When the subtype code is 1 (release records), a fourth word is used that contains the tape hookup time in seconds.

System Records

System accounting (type 5) records are entered into the accounting file when the system is autoloaded or recovered, or when new date or time information is entered. A system record consists of a four-word header, plus additional words depending on the record subtype. The header consists of four words: the first is as shown in figure 7-4, where the rightmost 40 bits are unused, and the subtype codes are 0 (autoload), 1 (recovery), 2 (new date entered), and 3 (new time entered); the second word is unused; the third word is the value of the master clock at the time this record was created; and the fourth word is unused.

When the subtype code is 0 (autoload records), four additional words are used:

- The first word contains the current date, eight ASCII characters in the format mm/dd/yy, where mm, dd, and yy are decimal numbers signifying the month, day, and year.
- The second word contains the current time, eight ASCII characters in the format hh:mm:ss, where hh, mm, and ss are decimal numbers signifying the hour, minute, and second.
- The third word contains the current central processor clock time in microseconds.
- The fourth word contains the number of small pages currently available.

When the subtype codes are 1 (recovery), 2 (new date entered), and 3 (new time entered), three additional words are used. The format of the additional words is the same as that of words 4, 5, and 6 for subtype 0, explained above.

Job Records

Job accounting (type 6) records are entered into the accounting file for the start and end of jobs. Job records consist of a four-word header plus one additional word. The format of the header is the same as for the task record header (figure 7-5), except that the type field is 6, and the subtype field can be 0 (job start) or 1 (job end).

A fourth word, which contains the job name for a batch job, is required for both subtype 0 (job start) and subtype 1 (job end) records. A batch job is defined as existing from the time that the start-of-job record is written until an end-of-job record is written by the batch processor.

STANDARDIZED ACCOUNTING ENHANCEMENTS

This section describes the set of Standardized Accounting Enhancements (SAE) for use on the CYBER 200 operating system. It includes the algorithm for computing System Billing Units (SBUs), and also describes the Variable Rate Accounting feature (VRA).

Accounting calculations are a part of the virtual system. At the end of each accounting period on the CYBER 200 operating system, the System Billing Units (SBUs) used for a task are computed and entered into accounting records.

CALCULATION OF SBUs

The formula for calculating SBUs is based on user CPU time and variable rate factor. The algorithm used in Standardized Accounting Enhancements is shown in figure 7-10.

```
SBU=PF(SL)*(VRF(VRI)*W(1)*SR(1)+SUM(W(I)*SR(I)))
```

where:

PF	is the weighting factor for priority or service level.
SL	is the service level index.
VRF	is the variable rate factor.
VRI	is the variable rate index.

Figure 7-10. Algorithm for SBU Calculation

(W(I),I=1,16) is the set of weighting factors associated with the set of system resources (SR(I),I=1,16). A set of installation parameters representing these factors is maintained by the system.

System resources are described as shown in table 7-4. The value for each system resource is determined by the system, dependent on the user task activity in the current accounting period.

VARIABLE RATE ACCOUNTING

Private controllee files running on the system have a standard rate at which the SBUs they consume during execution are charged. Certain other controllees, such as public utilities or applications packages, can be charged at a different rate whose relation to the standard rate is a fraction called a Variable Rate Factor (VRF).

The set of defined Variable Rate Factors (VRFs) is maintained in a virtual system table known as the Variable Rate Table, T_VRF. SAE makes provisions for the maintenance and use of the table. The set of Service Level Factors, which control job cost dependent on job class (priority, interactive, batch, or, standby), are also located in this table.

VARIABLE RATE TABLE

The Variable Rate and Service Level Factors are stored on a mass storage file, Q5VRF. The file size is one page. Each entry in this file is a non-negative floating point value. The first 256 words of the file are reserved for the Variable Rates. The following 64 words are reserved for Service Level Factors. A file entry set to 1.0 represents a variable rate equal to the standard installation charge. In

each word of the file, a single entry can be stored; entries need not be consecutive. It is suggested that the installation enter some default value such as 1.0 in location 0 of the file, as all system files will be initially created with a VRI setting of 0.

VIRTUAL SYSTEM TABLE DEFINITION

If an installation allows Variable Rate processing (IP_F_VR=1), the following steps are taken at Autoload time to define the Variable Rate Table, T_VRF. If the installation parameter IP_F_VR is set to 0, meaning that Variable Rates are not valid at this installation, the Variable Rate Table is never allocated, and no Variable Rate processing is done.

AUTOCON locates the PFI entry for the mass storage file which contains the set of entries for the Variable Rate Table.

The file name Q5VRF and the operator user number are used as defaults by AUTOCON. An alternate file can be specified by entering the line:

VR = file name , user number

at AUTOCON execution time. If the file is not found, AUTOCON displays the message:

VARIABLE RATE FILE NOT FOUND

and allows the operator to either correct the file name or continue without retrying.

The following system variable is added to the configuration table display:

55. Variable rate file VR=Q5VRF,999998

The following keywords are added to the keyword list obtained by typing "HELP":

56. Variable rate file VR=FILENAME,USERNO.

The file name Q5VRF and user number 999998 are installation parameters.

FILE MAINTENANCE

Q5VRF is maintained as a private file under the operator's user number. The utility LOOK can modify or display the contents of Q5VRF. If the file entries are changed, an autoload must be done to load the new values into the virtual systems Variable Rate Table.

TABLE 7-4. SYSTEM RESOURCES

I	SR(I)	Definition
1	UCPU	User execution CPU time (microseconds) used during this accounting period
2	SCPU	System CPU time (microseconds) used during this accounting period
3	MEMU	Memory usage during this accounting period: working set size * CPU time
4	LPACCX	Number of disk accesses (I/O requests issued for large page explicit reads and writes during this accounting period
5	LPACCI	Number of disk accesses (I/O requests issued for large page implicit writes during this accounting period
6	SPACCX	Number of disk accesses (I/O requests issued for small page explicit reads and writes during this accounting period
7	SPACCI	Number of disk accesses (I/O requests issued) for small page implicit writes during this accounting period
8	SPSECX	Number of disk sectors transferred for explicit reads and writes during this accounting period
9	SPSECI	Number of disk sectors transferred for implicit writes during this accounting period
10	LFGFLT	Number of large page faults (accesses, I/O requests issued) for faults during this accounting period
11	SPGFLT	Number of small page faults (accesses for faults) during this accounting period
12	TAPWDS	Number of 16-bit bytes transferred to or from tape during this accounting period
13	TAPACC	Number of tape acceses, one for each read or write during this accounting period
14	TAPFNT	Number of tape functions (other than read or write) during this accounting period
15	AVWSS	Average working set size during this accounting period
16	VSCALLS	Number of virtual system user calls made during this accounting period
17	LSPC	Number of large pages lost
18	LLPC	Number of small pages lost
19	WS2SML	CPU time (microseconds) that task's working set size limit appeared to be too small

This section contains information for a system programmer who is interested in developing an application or utility.

CONVENTIONS

A controllee execute line is entered for processing by CYBER 200 OS either as a batch processor control statement or as an interactive terminal type-in. An execute line can occur as one or more physical records representing card images or terminal lines. From the point of view of the common execute line supporting routines, an exact correspondence exists between batch commands and terminal commands, including continuation of the command text to more than one card or terminal line. (From the point of view of the user, however, this correspondence does not exist.)

Standard processing is done on five types of linguistic expressions called tokens. The tokens are:

- Alphanumeric identifiers.
- Decimal numeric constants.
- Hexadecimal numeric constants prefixed by the character #.
- Character or string constants delimited by the character ".
- The special characters, which are / # " & .) , = and blank.

The # character is referred to in text as a hash mark. The & character is an ampersand.

Execute line options are defined by means of positional or keyword-identified values. Standard diagnostics are issued if abnormal syntax or conditions are encountered.

A set of three system library routines are to be used to guarantee adherence to the conventions stated above. The routines are:

Q7KEYWRD Processes the text of an execute line.

Q7MODE Determines if the operation is in batch or interactive mode.

Q7PROMPT Provides interaction with the controller; collects parts from several input records, and builds the complete character string for processing by Q7KEYWRD.

When a controllee execute line requires more than one terminal line an ampersand must be used to designate continuation to the subsequent line. Card image continuation is performed automatically during batch processing if a terminator character has not been encountered. The ampersand signals a logical end-of-record and can be followed by comments. The text of the execute line consists of two or more tokens: the first is alphanumeric and identifies the task name, while the last is a special character called a terminator. The terminator characters are period and right parenthesis. An implicit terminator occurs at the end of a terminal line that does not contain an ampersand. Comments can be placed immediately following a terminator character or an ampersand. The following execute lines are equivalent:

```
SAMPLE,A.       optional comments

SAMPLE&,A.      optional comments

SAMPLE A
```

A parameter list can follow the task name but must precede the terminator character. Order-dependent parameters must be in the order specified; key-dependent parameters can appear in any order. Parameter formats depend on the control statement specified, but they always follow the same general guidelines.

Consecutive parameter list items are separated by level-1 separator characters comma and blank. In addition, the left parenthesis acts as a level-1 separator between the task name and the parameter list. A parameter list item can be defined by a list of user numbers or file names. These values also are separated by the level-1 parameter separators. A file name can be followed by attributes of disposition code or length, with attributes separated from each other by the level-2 separator character slash.

Blank is a special character and only performs a separator function when not used with other separators or terminators. Any level of separator can be preceded or followed by blanks, which serve only to highlight the separator; in a similar fashion, the terminator characters can be preceded by highlighting blanks.

System utilities or tasks provide default settings for all on/off options. In addition, the input, output, and binary file options have the default names INPUT, OUTPUT, and BINARY. Where tasks create files for the user, the task can determine the necessary file size or the user is allowed to submit an estimate of an adequate size. Tasks that create files also determine the disposition of the file upon task completion. The user has the opportunity to specify file disposition.

The task name is constructed of one to eight letters and digits. Except where reference is made to a drop file, the first character must be a letter. The task name is bound on the left by the start of the command and on the right by a level-1 separator or a terminator.

Order-dependent parameters are strings of nonseparator, nonterminator characters. Their interpretation is strictly a function of the particular product. An order-dependent parameter list is ended by a terminator or by the occurrence of a key-dependent parameter.

The following are examples of execute lines using order-dependent parameter lists:

```
COPY(FILEA,FILEB)

PURGE,FILE1,FILE2,FILE3.
```

A key-dependent parameter has the general structure shown in figure 8-1. The following is an example of an execute line using a key-dependent parameter list:

```
FTN(I=COMPILE,L=OUTPUT,B=BINARY/PU/#240)
```

key=defns

key	A string of letters and numbers, 1 to 255 characters, delimited on the left by a level-1 separator and on the right by an = character, a separator, or a terminator.
defns	Strings of nonseparator, nonterminator characters whose interpretation is strictly a function of the particular product and the key identifier.

Figure 8-1. Key-Dependent Parameter Format

Examples of the use of both parameter forms are:

```
WXYZ(FILE1,FILE2,OU=MAPFILE)
```

To ensure that ambiguities do not arise, the programmer calling the keyword word processors must not allow the following:

A parameter resembling a file name to follow a file name list unless that parameter has a key

A parameter resembling a user number to follow a user number list unless that parameter has a key

Parameter values can be strings of letters and digits, decimal digit strings, hexadecimal digit strings, and character strings delimited by quotation marks. In some cases, the alphanumeric string can occur as two decimal digits followed by one to six letters and numbers. This exception is provided to accommodate drop file names. Decimal digit strings are normally interpreted as decimal constants; a hexadecimal constant is normally preceded by a hash mark. Values that must be virtual bit addresses are always hexadecimal values even if the hash mark is not present. In some cases, such as the GROS option of the loader, a hash mark is required to distinguish the address from identifier data in the same list. Some examples are:

```
WXYZ(EN="!FILE!",OU=MAP/#10)

WXYZ(FILE,OU=MAP/16,LI=SYSLIB,MYLIB)

PURGE,12DROP,JUNK.

COPY,42DROP,SAVEDROP.
```

Lists of values are as order-flexible as the values permit; the user is normally given maximum flexibility consistent with the task requirements. The following equivalent parameter strings illustrate this flexibility:

```
B=FILE/10/PR

B=FILE/PR/10
```

All key-dependent parameters have on and off settings where appropriate, and can be turned on and off. Turning on keys can be accomplished by means of a key=1 parameter, or by use of the key name only; these keys can also be turned off by means of a key=0 parameter. File identification keys should be turned off with key=0. Where the option is normally off, a parameter of the form key=filename turns the option on for a specific file, while use of the key name only turns the option on for a default file.

The following execute lines are equivalent, and illustrate the on/off ability:

```
IMPL,X.

IMPL,I,X=1.

IMPL,X,I=INPUT,B=BINARY/#40.
```

SUPPORTING ROUTINES

Common execute line standards are supported by three subroutines from the system library. The subroutines, which are callable from FORTRAN, META, and IMPL, are:

Q7MODE	Determines whether the parent controller of the calling task is a terminal or another task. A batch job falls into the latter category.
Q7PROMPT	Inputs parameters to be passed to Q7KEYWRD for syntax checking. It prompts terminal users for input if no parameters are specified in the execute line. It also strips the trailing period or matching outside parenthesis characters from the parameter text before calling Q7KEYWRD.
Q7KEYWRD	Examines a character string, checks its syntax, and converts data to internal format. In the case of a detected error, it prints error messages and requests; in interactive mode, it permits error correction by accepting reinput of an execute line parameter. Also, in interactive mode, Q7KEYWRD can be set to request and input each parametric keyword through the use of an appropriate prompting message.

Assembly language routines call any of these subroutines by using the FORTRAN or IMPL type of calling sequence, while FORTRAN and IMPL programs access the routines using CALL statements.

If the main program is coded in FORTRAN, the original execute line is processed by FORTRAN initialization for run-time file substitution. In interactive mode, the program may subsequently call Q7PROMPT or Q7KEYWRD for other lines.

Q7MODE

The function of this subroutine is to determine program environment and to return the information in a full word whose variable name is supplied as the only parameter to the Q7MODE routine. A full word is defined as a 64-bit word that is aligned on a word boundary. The call statement format of Q7MODE is shown in figure 8-2.

CALL Q7MODE(mode)

mode — A full-word variable in which one of the following values is returned:

- 0 Controller is not a terminal
- 1 Controller is a terminal

Figure 8-2. Q7MODE Call Statement Parameters

Q7PROMPT

This routine serves as an interface between a calling routine and the Q7KEYWRD subroutine. It inputs user parameters into an input buffer, then passes the text to Q7KEYWRD for syntax checking. If no text is specified on the execute line, Q7PROMPT can prompt the interactive user for parameters, using the message PLEASE SPECIFY PARAMETERS or a message provided by the calling routine; otherwise, if no text is specified on the execute line, it can proceed with a call to Q7KEYWRD, optionally setting bit 60 in the options parameter, which causes Q7KEYWRD to prompt for individual keywords.

An input buffer can either be supplied by the calling routine or allocated by Q7PROMPT. Delineator characters, such as matching outside parentheses or a trailing period, are deleted from the input text prior to the call to Q7KEYWRD. Input text can be continued on succeeding lines in interactive mode, provided that an ampersand (continuation character) is appended to each line.

The call statement format of Q7PROMPT is shown in figure 8-3. The opt, r, rbuf, rlen, and t_i parameters are not used by Q7PROMPT, but are passed to Q7KEYWRD for use in syntax checking. If a text string is not specified in the controllee execute line and the p parameter is nonnegative, Q7PROMPT prompts for parameters and saves them in a buffer with the name specified as the buf parameter.

CALL Q7PROMPT(txt,p,opt,r,buf,blen,rbuf,rlen,t_1,. . .,t_n)

txt — Text string to be passed to Q7KEYWRD. The string must contain any desired carriage control characters.

p — Indicates whether prompting is desired:

- >0 Number of character bytes in txt. Use txt to prompt for parameters.
- 0 Use the text string PLEASE SPECIFY PARAMETERS to prompt for parameters.
- -1 Do not prompt for parameters. The value of the variable blen is zero.
- -2 Do not prompt for parameters. Options bit 60 should be zero.
- -3 Do not prompt for parameters. Wait for message.

buf — Name of buffer file into which the parameters are to be read. If the blen field is zero, the buf field is the name given to a buffer provided by Q7PROMPT.

blen — Name of a full-word variable whose nonzero value indicates the number of character bytes in buf. If the value of blen is zero, no buffer is provided by the caller; in this case, Q7PROMPT allocates a 4096-character buffer named buf. A count of the number of characters actually read is returned by the system into the blen field.

The opt, r, rbuf, rlen, and t_i fields are described under the Q7KEYWRD call statement.

Figure 8-3. Q7PROMPT Call Statement Parameters

Q7KEYWRD

The keyword subroutine scans a line of text, checks syntax, and converts data to internal formats. It prints error messages and inputs replacement expressions as required. Q7KEYWRD processes text containing both positional and keyword type parameters. The calling routine provides Q7KEYWRD with syntax tables that completely describe the general format of the input parameters. Q7KEYWRD uses the tables to interpret the specific parameters in the execute line test. These input parameters, called keyword expressions, are written as follows:

key_1 key_2 key_3 . . .

Each key_i is separated from other keyword expressions by one or more blanks or by commas, and has one of the following formats:

lhs = rhs

lhs

rhs

The syntax tables for each key_i keyword relate to the valid left-hand sides (lhs) of the expression and valid right-hand sides (rhs) for each corresponding left-hand side. This includes specifying whether the degenerate cases, lhs and rhs, are to be treated as having no left-hand side or no right-hand side. Each key_i, then, can be any one of the following possibilities:

lhs(1) = rhs(1,1)
.
.
.
lhs(1) = rhs(1,n1)
lhs(2) = rhs(2,1)
.
.
.
lhs(m) = rhs(m,nm)

where m is the number of possible left-hand sides for the expression, left-hand side k having nk possible right-hand sides.

The syntax tables also specify positional relationships among the keyword expressions. A given expression, key_i, can be flagged as positional, meaning that it must appear after expressions $key_1, \ldots, key_{(i-1)}$ but before the expressions $key_{(i+1)} \ldots$ If an expression (key_i) is not flagged as positional, it can appear in any order with preceding or succeeding nonpositional expressions; so, if key_i, $key_{(i+1)}$, and $key_{(i+2)}$ are nonpositional, any of the following are valid:

key(i), key(i+1), key(i+2)
key(i), key(i+2), key(i+1)
key(i+1), key(i), key(i+2)
key(i+1), key(i+2), key(i)
key(i+2), key(i), key(i+1)
key(i+2), key(i+1), key(i)

The syntax tables also relate which key_i parameters are required in the execute line text. If a parameter flagged as required is not encountered in its required location, an error message is issued.

Left-hand sides for an expression include:

- None (the degenerate case, lhs).
- A literal character string, 1 to 255 characters long.

Right-hand sides for an expression include:

- None (the degenerate case, rhs).
- A literal character string, 1 to 255 characters long.
- An arbitrary character string 1 to 255 characters long.
- Any remaining unscanned text, up to 255 characters maximum.
- A number in the range 0 to $2^{47}-1$ (table setting indicates whether the number can be decimal, hexadecimal with a leading # character, or an address in hexadecimal with no leading # sign required; table settings can also indicate the range of the number if the default range is not sufficiently restrictive).
- A user number.
- A file name (table settings indicate whether a drop file name can be specified and whether the length, print, and punch attributes can be specified).

Lists of numbers, user numbers, and file names separated by slashes, blanks, or commas can be allowed as right-hand sides. A field in the syntax tables indicates that lists are to be allowed and specifies the maximum number of elements permissible.

The entry point Q7KEYWRD is used for both FORTRAN and IMPL calling sequences. The call statement format of Q7KEYWRD is shown in figure 8-4. The lhs table pointers are illustrated in figure 8-5. Each entry in an lhs table points to an rhs table (also full-word-aligned) that describes valid right-hand sides for the given left-hand side and specifies the format in which information is returned in the return buffer to the calling routine.

lhs Table

An lhs table consists of contiguous, variable-length, full-word-aligned entries describing valid keyword expressions. The entries describe the left-hand sides of expressions and, in turn, point to tables whose entries describe valid right-hand sides. A header relates positional and existence requirements of the keywords described by this table.

The lhs table format is shown in figure 8-6. The table header contains two words in the format shown in figure 8-7. Each lhs entry has the format shown in figure 8-8.

rhs Table

The rhs table contains contiguous, variable-length, full-word-aligned entries that describe valid right-hand side expressions. The table format is shown in figure 8-9. The first word of each rhs table entry has the format shown in figure 8-10.

When the type field is zero, the rhs table entry is one word having the format shown in figure 8-10, but with the right, flags, and count fields unused. When the type field is 1, the rhs table entry is one word having the format shown in figure 8-10, but with the flags and count fields unused. When the type field is 2, the format of the rhs table entry is as shown in figure 8-11.

When the type field is 4, the format of the rhs table entry is as shown in figure 8-10, except that the flags and count fields are not used. The right-hand side of the expression (255 characters maximum) is placed in the return buffer without further validation or conversion. Enclosing quotes are required only if a special character is part of the text.

When the type field is 5, the format of the rhs table entry is as shown in figure 8-10, except that the flags and count fields are not used. When the type field is 6, the format of the rhs table entry is as shown in figure 8-12.

When the type field is 7, the format of the rhs table entry is as shown in figure 8-10. Four of the individual bits in the flags field can be set to 1, in which case they have the following meanings:

Bit	Meaning
20	Punch attribute (PU) is valid.
21	Print attribute (PR) is valid.
22	Length attribute can be specified.
23	Drop file name can be specified.

Two or more of the bits can be set at one time. The count field contains the number of file names that appear in the associated return buffer entry.

CALL Q7KEYWRD(opt,r,buf,blen,rbuf,rlen,t_1, . . . ,t_n)

opt — Name of a full-word variable, the rightmost four bits of whose value indicate the following options:

Bit	Meaning
60	If zero, scan the input for keyword expressions. If set to one, prompt for each keyword listed in the tables t_1, . . . ,t_n.
61	If zero, abort on cancel response or on syntax error. If set to one, return to caller on either condition.
62	If zero, prompt for replacement on syntax error. If set to one, do not prompt for replacement.
63	If zero, send error messages to program controller for output. If set to one, do not output error messages.

r — Name of full-word variable to contain return codes. Return codes are:

0	Text scanned successfully
1	Error or cancel response
2	Return buffer too small
3	Incorrect number of parameters
4	Invalid type field in lhs table entry
5	Invalid type field in rhs table entry
6	Invalid flags field in rhs table entry
7	Words field for return buffer entry exceeds 255
8	Options field bit 60 is 1, and prompt message length or address in lhs table header is zero

Code 1 is returned only if bit 61 of opt field is 1.

buf — Virtual bit address of string to be scanned for keyword expressions. This field is not used if prompting is requested (options bit 60 is 1).

blen — Name of full-word variable whose value specifies the number of characters in the string indicated by buf. This field is not used if options bit 60 is 1.

rbuf — Virtual bit address of the full-word-aligned buffer (the return buffer) in which reformatted keyword information is to be returned.

rlen — Name of full-word variable whose value specifies the number of characters in the return buffer.

t_i — Virtual bit address of full-word-aligned lhs table (figure 8-5) that describes acceptable syntax constructs and specifies formats for the returned information. The number of addresses varies with the syntax of the line being scanned.

Figure 8-4. Q7KEYWRD Call Statement Parameters

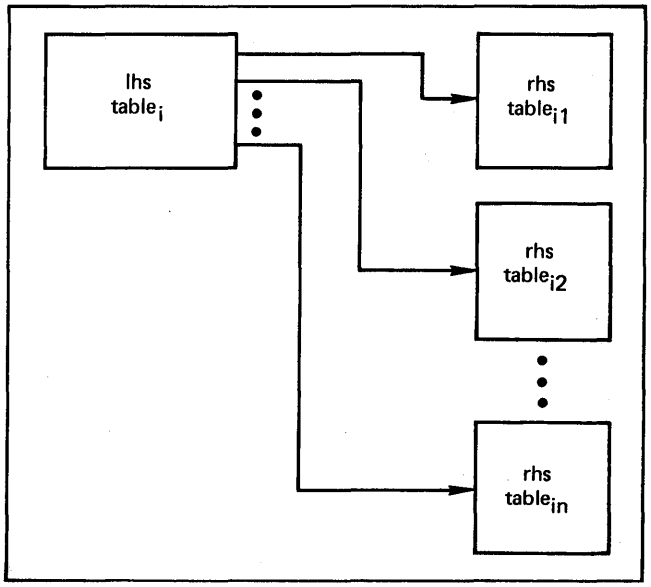


Figure 8-5. lhs Table Pointer Configuration

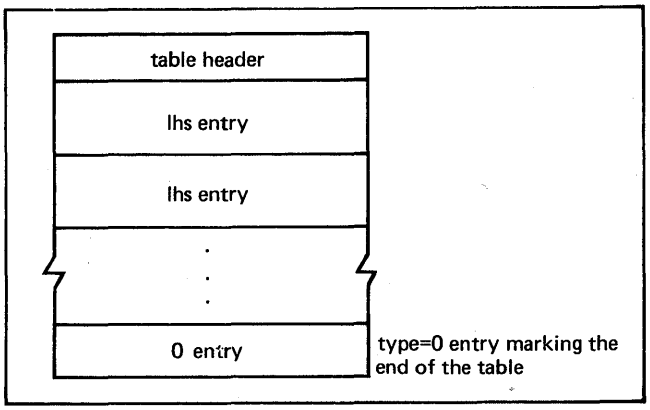


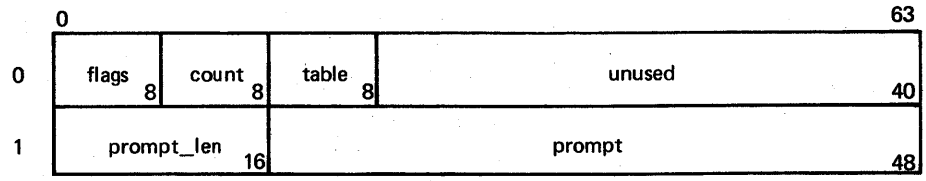
Figure 8-6. lhs Table Format

When the type field is 8, the format of the rhs table entry is as shown in figure 8-10. Two of the individual bits in the flags field can be set to 1, in which case they have the following meanings:

Bit	Meaning
22	Return an ASCII value.
23	Return a binary value.

One or both of the bits can be set at one time.

When the type field is 9, the format of the rhs table entry is as shown in figure 8-10, except that the right, flags, and count fields are not used. For this type, no entry is made in the return buffer and processing continues with the next keyword expression.



flags — Bits that are set to describe keywords:

- 6 — If zero, entries describe a keyword that is not positional (that is, the keyword described can appear in any order with preceding or succeeding nonpositional keywords); if set to 1, entries describe a positional keyword.
- 7 — If zero, entries describe an optional keyword; if set to 1, entries describe a required keyword (if no match is found, an error message is issued).

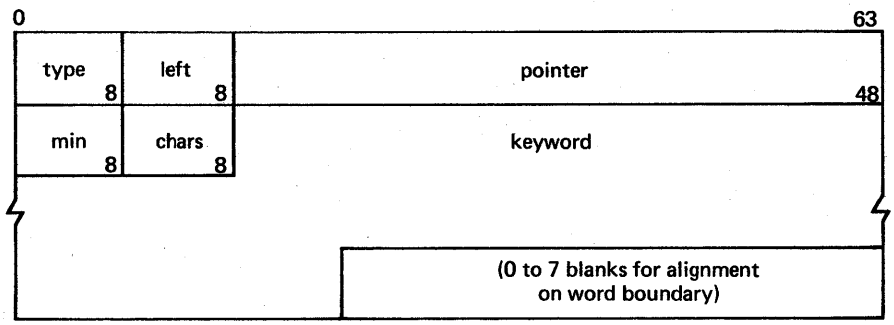
count — A value that specifies the maximum number of times this table can be used to effect a keyword match.

table — A value set by the caller and returned in a return buffer entry on a successful lhs and rhs match. (The return buffer is described later in this section.)

prompt_len — A value that specifies the number of characters in a message whose address is given in the prompt field; valid only when the options bit 60 is set to 1.

prompt — Address of the text to be output as a prompt to request keywords associated with this table; valid only when the options bit 60 is set to 1. Any ASCII carriage control characters desired must be embedded in the text of the prompting message.

Figure 8-7. lhs Table Header Format



type — Entry type. The values are:

- 0 — End of the table
- 1 — There are no left-hand sides
- 2 — The keyword expression contains a left-hand side

left — A value set by the caller and returned in the return buffer entry upon a successful left-hand side and right-hand side match.

pointer — Address of the table describing right-hand sides that are valid with this particular left-hand side.

min — Minimum number of characters needed in the left-hand side before attempting a substring match against the keyword (type=2).

chars — The number of characters in the keyword if the type field is 2; must be zero if the type is 1.

keyword — Text to be used in validating the left-hand side of the expression (type=2).

Figure 8-8. lhs Table Entry Format

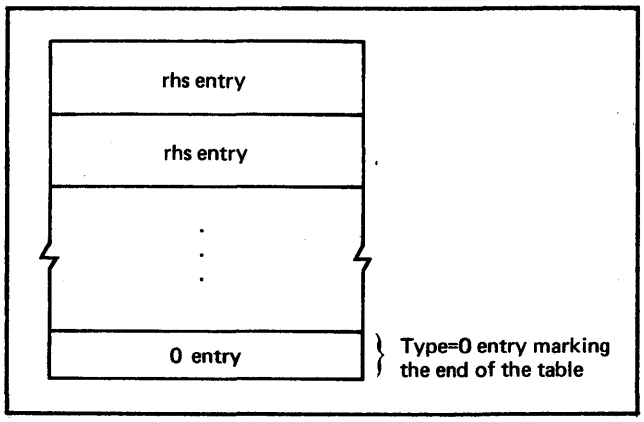
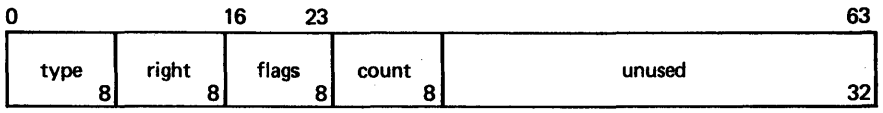


Figure 8-9. rhs Table Format



type — Entry type number. The format of each entry and the meaning of its flags and count fields vary according to the entry type. The types are:

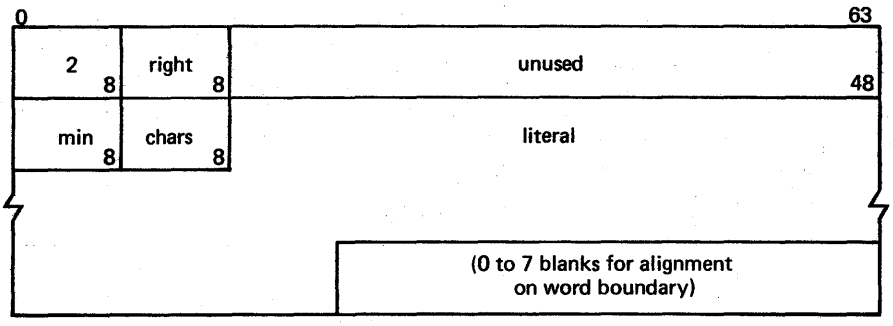
0 End of the table
1 No right-hand side in the expression
2 Literal; the right-hand side of the expression must match the initial substring of the literal
3 (not used)
4 Arbitrary character string is returned in the return buffer
5 All remaining text (255 characters maximum) is returned in the return buffer
6 Numbers in the range of 0 to $2^{47}-1$ are returned in the return buffer
7 File names are returned in the return buffer
8 User number
9 Ignore the keyword

right — A value set by the caller and returned in the return buffer entry on a successful left-hand side and right-hand side match.

flags — Flag bits, which are set to describe valid right-hand sides of expressions. Bit meanings depend on entry type.

count — Maximum number of elements in the right-hand side for those entry types that allow lists.

Figure 8-10. General Format for First Word of rhs Table Entry

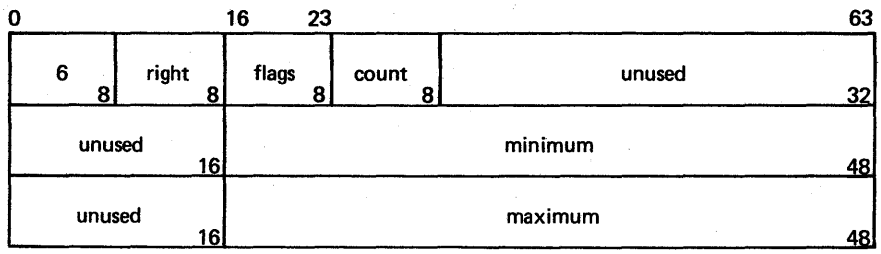


min — Minimum number of characters needed in the right-hand side before attempting a substring match against the literal.

chars — Number of characters in the literal.

literal — Text to be used in validating the right-hand side of the expression.

Figure 8-11. rhs Table Entry Format, Type 2



flags — Flag bits. If bit 20 is zero, the second and third words are not present. When set to 1, bits 20 through 23 select the following options:

20	Range check is desired; the number specified must be greater than or equal to minimum and less than or equal to maximum
21	Hexadecimal number valid with a leading hash mark
22	Address valid (hexadecimal number without a leading hash mark)
23	Decimal number valid

Two or more of bits 21, 22, and 23 can be set at one time; however, a potential identification problem exists if bits 22 and 23 are both set: if the field contains digits 0 to 9 only, the number is treated as decimal, and otherwise, it is assumed to be hexadecimal.

count — Maximum number of values that can appear in the right-hand side.

minimum — Lower boundary for a valid number.

maximum — Upper boundary for a valid number.

Figure 8-12. rhs Table Entry Format, Type 6

Return Buffer

This buffer is used to contain reformatted keyword information that is returned. The end of the returned information is indicated by a full-word binary zero. The return buffer format is shown in figure 8-13.

The format and length of each return buffer entry depends on the type field of the right-hand side table entry that successfully matched the right-hand side of the keyword expression. Common to all entries is the first full word, whose format is shown in figure 8-14.

When the rhs table entry types are 1 and 2, the format of the return buffer entry is as shown in figure 8-15. The words field is always 1.

When the rhs table entry types are 4 and 5, the format of the return buffer entry is as shown in figure 8-16.

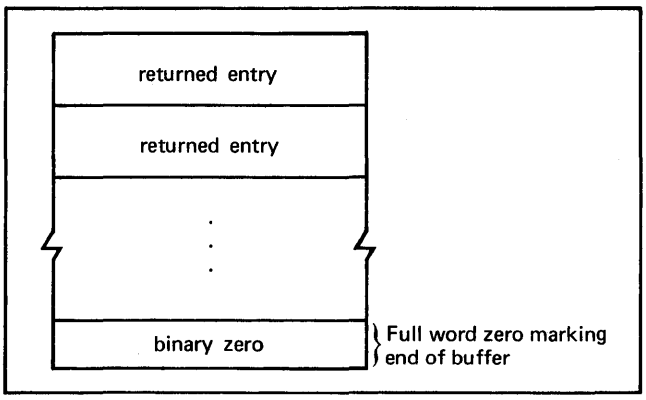


Figure 8-13. Return Buffer Format

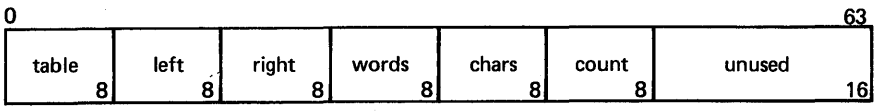


table	Value from the header of the lhs table (t_i) that provided the left-hand side and right-hand side entries effecting the keyword match.
left,right	Values from the left field of the particular lhs table entry and the right field of the rhs table entry effecting the left-hand side and right-hand side matches.
words	Total number of words in this particular entry.
chars	The meaning of this field varies with the type of the right-hand side. It is the length of the returned information or the length of an element of a returned list, such as a list of file names.
count	Number of returned elements if returned data consists of a list of items.

Figure 8-14. General Format for First Word of Return Buffer Entry

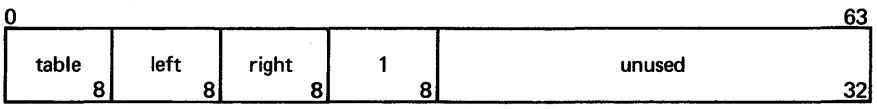
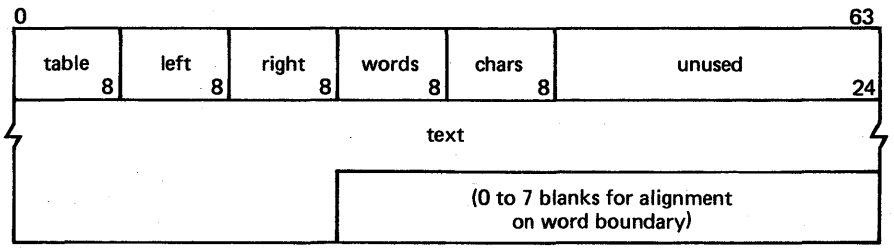


Figure 8-15. Return Buffer Entry Format, Types 1 and 2



chars	Number of characters returned in text.
text	The right-hand side of the expression (type=4), or the remaining text (type=5).

Figure 8-16. Return Buffer Entry Format, Types 4 and 5

When the rhs table entry type is 6, the format of the return buffer entry is as shown in figure 8-17. The chars field is always 8.

When the rhs table entry type is 7 with flag bits 20, 21, and 22 all set to zero, the format of the return buffer entry is as shown in figure 8-18. The chars field is always 8.

When the rhs table entry type is 7 with flag bits 20, 21, or 22 set to 1, the format of the return buffer entry is as shown in figure 8-19. The chars field is always 16.

When the rhs table entry type is 8 with only one of flag bits 22 and 23 set, the format of the return buffer entry is as shown in figure 8-20. The chars field is always 8.

When the rhs table entry type is 8 with both flag bits 22 and 23 set, the format of the return buffer entry is as shown in figure 8-21. The chars field is always 16.

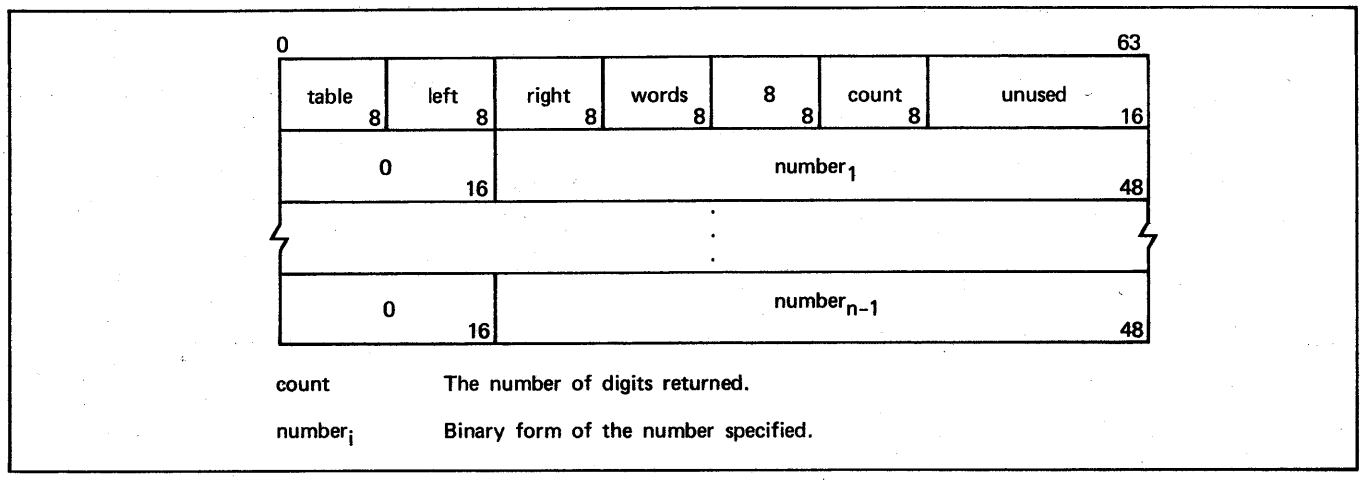


Figure 8-17. Return Buffer Entry Format, Type 6

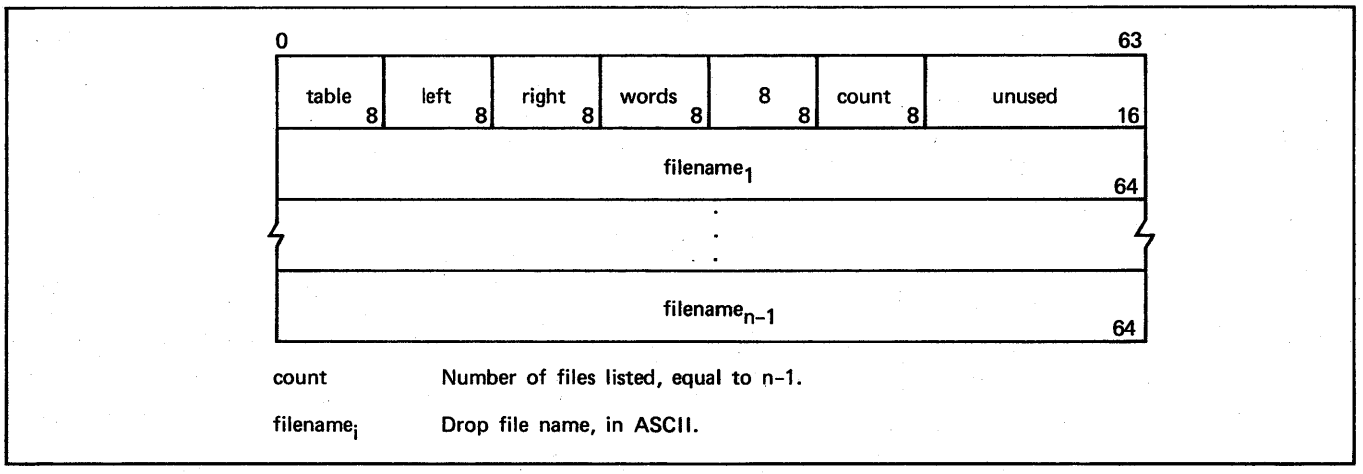
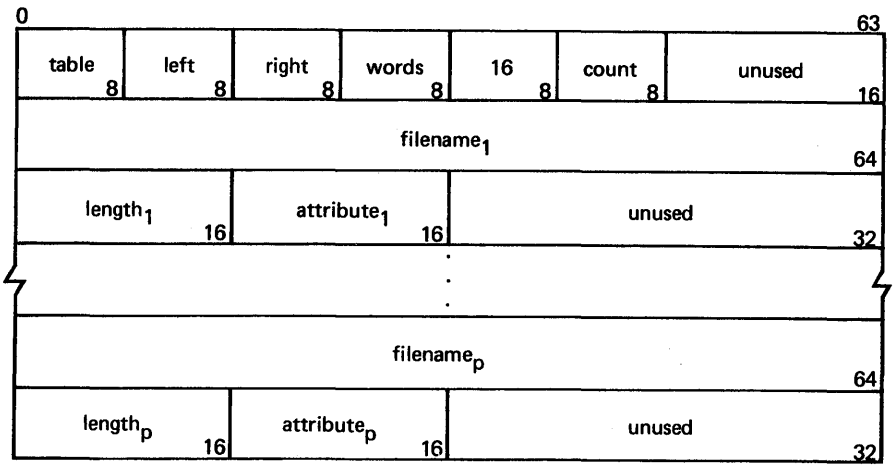
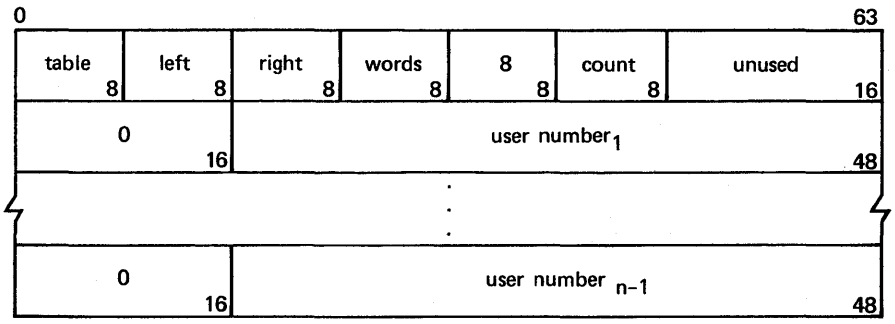


Figure 8-18. Return Buffer Entry Format, Type 7 with Zeroed Flags



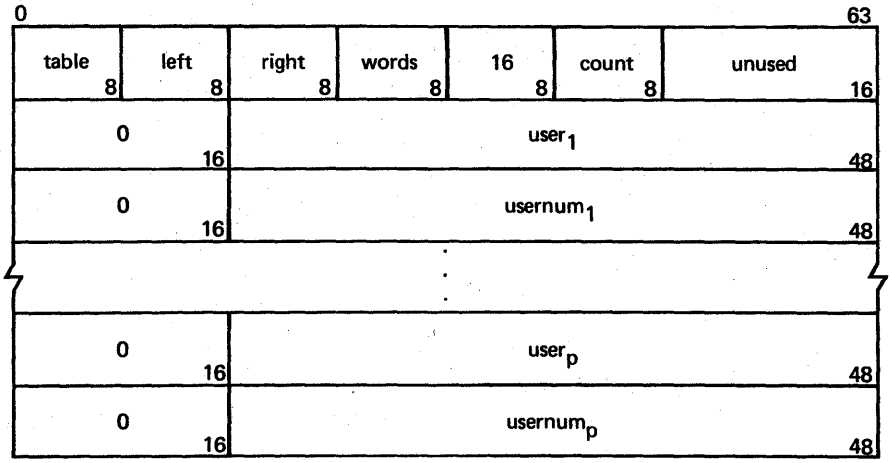
count	Number of files listed; has the value (n-1)/2.
$filename_i$	Name of the file specified, left-justified with blank fill.
$length_i$	Length of the file in small pages. If not specified, binary zero is returned.
$attribute_i$	File attribute: ASCII punch (PU) or print (PR). If not specified, blanks are returned.

Figure 8-19. Return Buffer Entry Format,Type 7 with Set Flag



count	Number of user numbers listed.
user number	ASCII or binary user number, depending on the flags set.

Figure 8-20. Return Buffer Entry Format, Type 8 with One Set Flag



count Number of user numbers returned, equal to (n-1)/2.

$user_i$ Binary user number.

$usernum_i$ User number of the user; interpreted as ASCII characters.

Figure 8-21. Return Buffer Entry Format, Type 8 with Two Set Flags

TABLE 8-1. EXECUTE LINE SPECIAL CHARACTERS

Character	Meaning
"	Delimits a literal character string on the right-hand side of the expression. The entire right-hand side must be included within the string (that is, Q7KEYWRD will not perform concatenation of a literal in quotes and other character strings). The delimiting quotes are not used for right-hand-side processing.
blank	Delimits a keyword expression unless it occurs within a literal character string.
,	Delimits a keyword expression unless it occurs within a literal character string.
=	Separates the left-hand sides and right-hand sides of keyword expressions unless they occur within a literal character string.

Special Characters

The Q7KEYWRD subroutine scans for special characters in the execute line text so as to extract keyword expressions. These characters and their meanings under given conditions are described in table 8-1.

This section contains formats for the following loader tables:

Module header table
Code block table
Code relocation table
External/entry table
Interpretive data initialization table
Interpretive relocation initialization table
Transfer symbol table
Debug symbol table
Symbol definition table
Pseudo address vector table

The following loader tables are used by the system during error processing:

Module header table
Code block table
External/entry table
Debug symbol table
Symbol definition table
Pseudo address vector table

Error processing information is provided for every object module loaded to produce a controllee file. This includes object modules of user-specified files and required object modules for system library files.

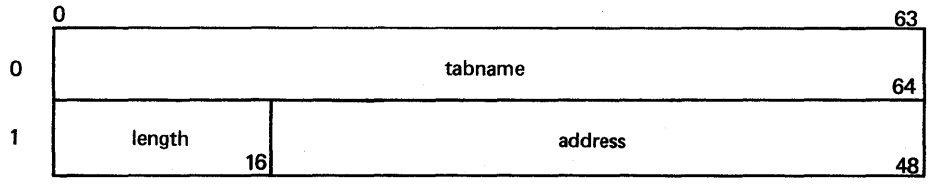
The loader initializes the following registers in the zero page of the controllee:

#06	Entry address - origin+8000
#0A	Version (from VR parameter)
#0B	Date
#0C	Time
#0D	Length and address of the error processing information
#14	Constant #20
#15	Constant #1A
#16	Constant 1
#1B	Dynamic stack address
#1C	Current register save area descriptor (length=6, address=dynamic stack address - #180)
#1E	Length and address of main data base
#1F	Entry address

Other registers are initialized to zero, but can be initialized to other values, as necessary.

GENERAL TABLE STRUCTURE

The loader works with files that are composed of one or more object modules. Each object module consists of a number of standard tables; each table begins with a standard two-word header. The format of the table header is shown in figure 9-1.



tabname — Name of the table, in ASCII. The table names are:

ΔMODULEΔ	INTΔRELO
ΔΔCODEΔΔ	XFERΔSYM
RELΔCODE	SYMBΔTAB
EXTΔENTR	SYMBΔDEF
INTΔDATA	ΔΔPAVΔΔΔ

length — Length of the table, in full words. For the debug symbol table, this field includes the length of the symbol definition table.

address — Bit difference between the first word of the respective table and word 1 of the module header table; that is, back pointer (bits) + address of the first word of the respective table (bits) = address of word 1 of the header table (bits).

Figure 9-1. Loader Table Header Format

MODULE TABLES

The module tables described here are the header, code block, code relocation, and external/entry tables.

MODULE HEADER TABLE

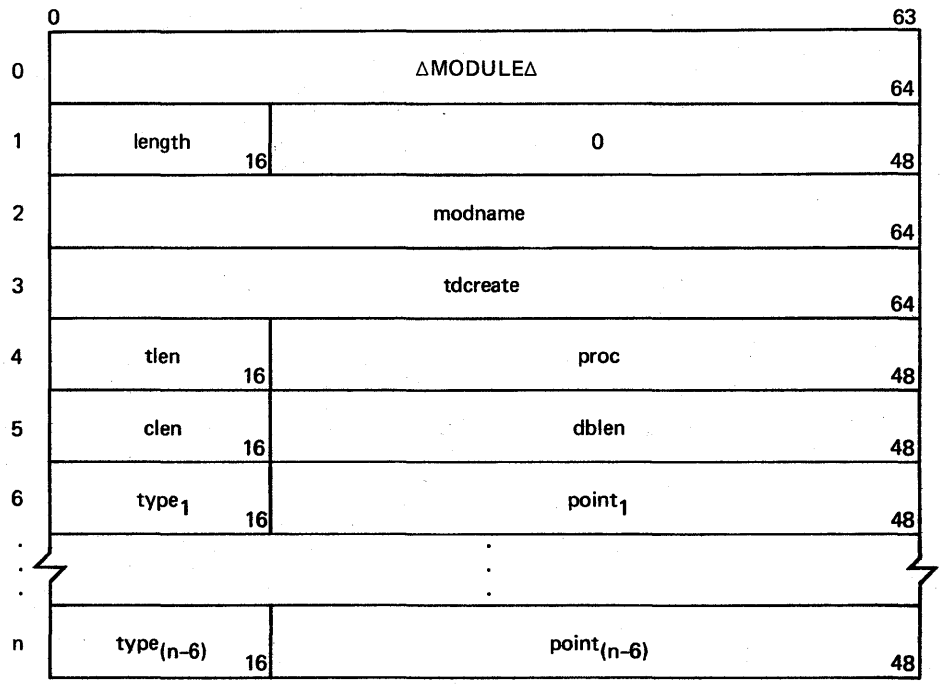
The module header table contains general information concerning the object module, and provides a linkage to all the other tables in the module. The format of the module header table is shown in figure 9-2.

Each of words 7 through n of the module header contains a table type and an address pointer to a table of that type. The pointer contains a bit address relative to the first word address of the header. By convention, the first table described is the code block table, and the second is the external/entry table.

Table types are listed in table 9-1. Only types 1, 2, 6, and 301 appear in the error processing information area of an object module.

CODE BLOCK TABLE

The code block table contains the executable code. The table consists of the two-word loader table header (figure 9-1), followed immediately by one or more words of executable code. The table name is ΔΔCODEΔΔ. The code block table has a pointer in the error processing information area. In this capacity, the table contains the program name (in ASCII) in word 1 rather than the character string ΔΔCODEΔΔ.



modname	Name of the module, in ASCII; eight characters, left-justified with blank fill.
tdcreate	Date and time the module was created; 15 decimal digits (in hexadecimal form) and a positive sign. The format is +yymmddhhttsscc, where yy expresses the year, mm the month, dd the day, hh the hour, tt the minute, ss the second, and ccc the millisecond.
tlen	Word length of tables, excluding the code.
proc	ASCII name of the processor that created the module.
clen	Length of the code, in words.
dblen	Length of the data base area, in bits. The maximum size of the data base is one large page.
$type_i$	Table type (see table 9-1).
$point_i$	Address pointer to a table of the type indicated in the type field. If the hexadecimal type is 4, the pointer contains the bit address of the next module header table.

Figure 9-2. Module Header Table Format

TABLE 9-1. MODULE HEADER TABLE TYPES

Type	Module Name	Description
1†	CODE	Code block table
2†	EXT ENTR	External/entry table
3	REL CODE	Code relocation table
5	XFER SYM	Transfer symbol table
6	SYMB TAB	Debug symbol table
10	INT DATA	Interpretive data initialization table
201	INT RELO	Interpretive relocation initialization table
301	PAV	Pseudo address vector table

† These types appear in the error processing information area of an object module.

CODE RELOCATION TABLE

This table describes relocation in the code. The format of the code relocation table is shown in figure 9-3. When this table is processed, the bit base address of the code is added to the 48-bit fields pointed to by the indexes in the bit string.

EXTERNAL/ENTRY TABLE

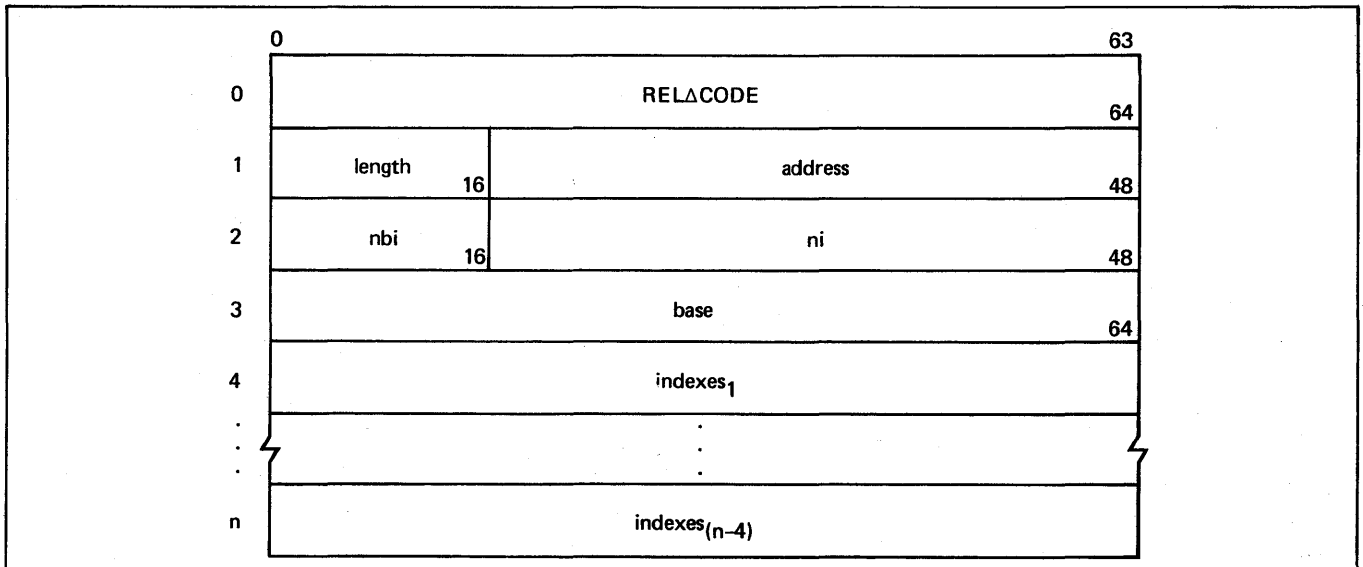
The external/entry table contains definitions for all entry points, external symbols, and common blocks. These definitions consist of lists of entry point names, external names, entry point descriptors, and external descriptors. The format of the table is shown in figure 9-4. In words 3+n and 3+2n, the quantity n-m is the number of external names in the table.

Each descriptor in the external/entry table has the form shown in figure 9-5.

The symbol types are defined as follows:

Entry point

A named value defined in the procedure; it is to be referenced as an external by an external procedure. It can be an address in the code block, an address in the data base, or a constant value.



nbi — Number of bits per index in the bit strings that start in word 5.

ni — Number of indexes in the string.

base — Current bit address to which this module is relocated.

$indexes_i$ — Bit string of indexes, each nbi bits long. Each index references a half-word of code to be relocated relative to the base address of the code.

Figure 9-3. Code Relocation Table Format

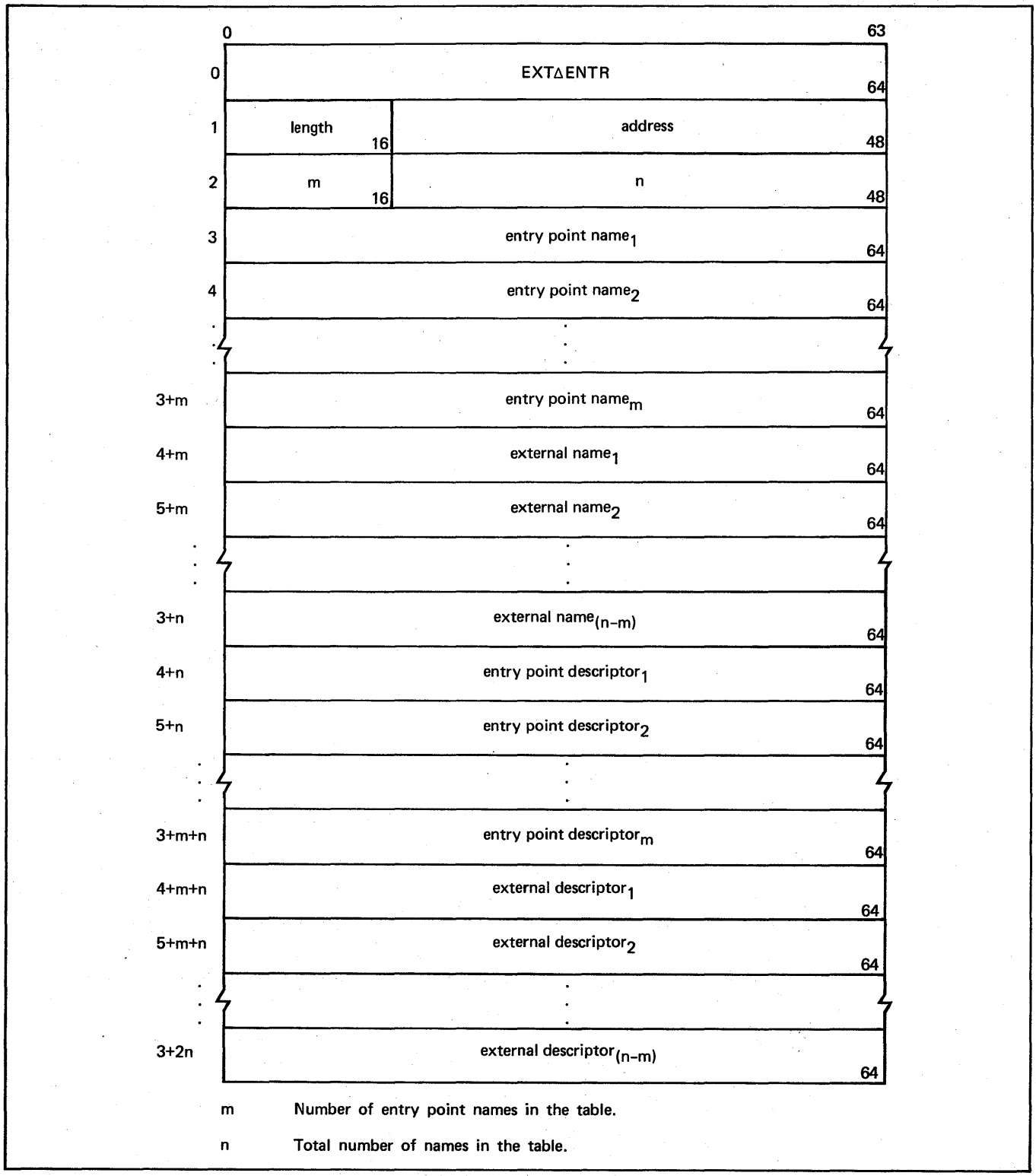
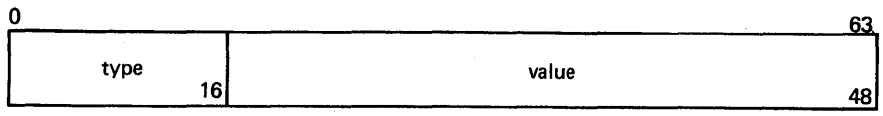


Figure 9-4. External/Entry Table Format



type Symbol type:

1	Entry point in code	14	External procedure
2	Entry point in data	15	External data
3	Constant entry point	16	Common block

value Depends on the type field. If type is 14 or 15, value is zero; otherwise, if type is 1 or 2, value is the entry point's relative bit address in the code or data block; if type is 3, value is a 48-bit constant; and if type is 16, value is the bit length of the common block.

Figure 9-5. Descriptor Format for Externals and Entry Points

Common block

A named alterable space referenced by one or more procedures. A common block can be initialized with relocatable data. Blank common is a common block with a name of eight blanks.

External procedure

An external that is referenced in a call. Having a symbol doubly defined as a common block and external procedure is specifically allowed. All external procedure names are eight characters, left-justified with blank fill.

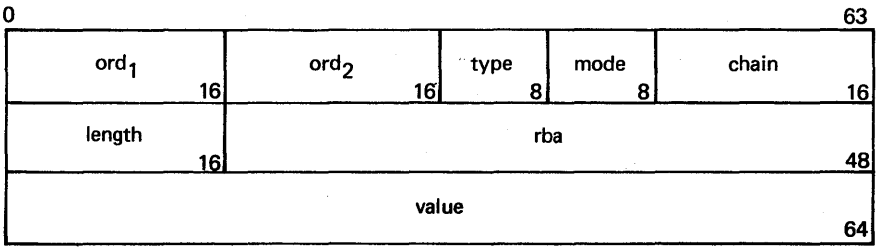
External data

An external that is referenced by a method other than a procedure call.

INTERPRETIVE DATA INITIALIZATION TABLE

When the loader processes information in the interpretive data initialization table, areas of static space are initialized. The table consists of the two-word loader table header (figure 9-1), followed immediately by one or more variable-length entries. The table name is INT DATA. Each entry contains a 1-word descriptor and 2-, 3-, 4-, or 6-word data item.

Data item and item descriptor pairs in the interpretive data initialization table are formatted as shown in figures 9-6 through 9-9. The first word in each figure is the data item descriptor. The remainder of each figure describes the data item proper, which is stored in the formats shown.



ord_1 Pseudo address vector table ordinal of static space to be initialized.

ord_2 Pseudo address vector table ordinal of space relative to which relocation is to be performed (relocation base).

type Type of data item that follows:

1	Full-word broadcast
2	Half-word broadcast (not defined if the mode field is 1)
3	Full-word vector transmit
4	Half-word vector transmit (not defined if the mode field is 1)
9	Byte string
A	Bit string

mode Mode flag:

0	Value to destination
1	Value plus relocation base to destination
2	Destination plus relocation base to destination

Figure 9-6. Data Item Format 1 (Sheet 1 of 2)

When the mode flag is 0, the values in the item are stored directly into the destination field (specified by the ord_1 field), and the ord_2 field is ignored. When the mode flag is 1, the relocation base is added to the values before they are stored in the destination field; for this case, half-word values are not defined. When the mode flag is 2, the relocation base is added to the destination field; in this case, the value field is absent in the data item.

chain — Full-word count to the next data item descriptor in the table (same as a count of the number of full words in the data item).

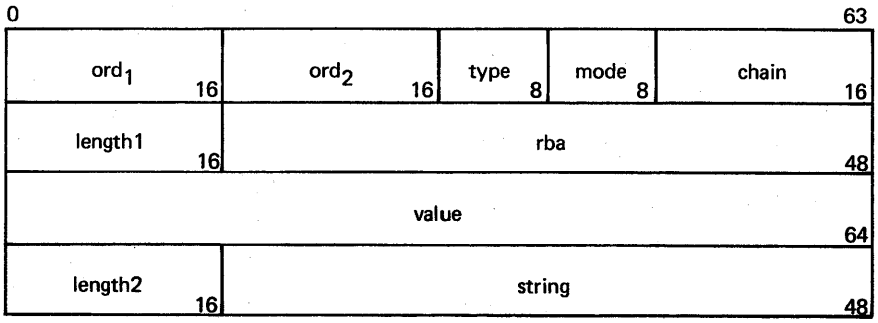
length — Bit length of the vector, for data item types 1, 2, 3, and 4; the number of bytes of information in the value field, for data item type 9; and the number of bits of information in the value field, for data item type A.

rba — Relative bit address.

value — A string or a vector value, depending on the data item type, as follows:

A full word to be stored in consecutive full words, starting at the relative bit address in the rba field (type 1)
A left-justified half-word to be stored in consecutive half-word locations, starting at the relative bit address in the rba field (type 2)
A full-word vector to be transmitted to the relative bit address in the rba field (type 3)
A half-word vector to be transmitted to the relative bit address in the rba field (type 4)
A left-justified byte string to be stored at the address in the rba field (type 9)
A left-justified bit string to be stored at the address in the rba field (type A)

Figure 9-6. Data Item Format 1 (Sheet 2 of 2)



ord_1 — Pseudo address vector table ordinal of static space to be initialized.

ord_2 — Pseudo address vector table ordinal of space relative to which relocation is to be performed (relocation base).

type — Type of data item that follows:

5 Full-word sparse vector
6 Half-word sparse vector (not defined if the mode field is 1)

mode — Mode flag:

0 Value to destination
1 Value plus relocation base to destination
2 Destination plus relocation base to destination

The meaning of this field is the same as that of the mode field in figure 9-6.

chain — Full-word count to the next data item descriptor in the table (same as a count of the number of full words in the data item).

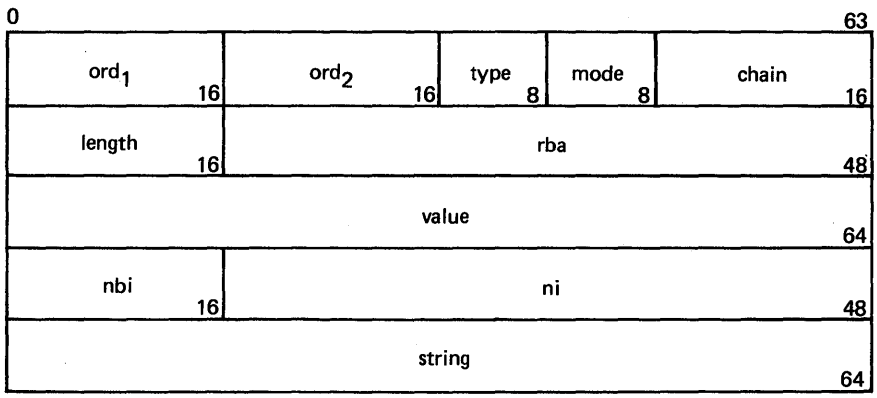
length1 — Number of 1-bits in the order vector specified in the string field.

rba — Relative bit address of the location to which the sparse vector is to be transmitted.

Figure 9-7. Data Item Format 2 (Sheet 1 of 2)

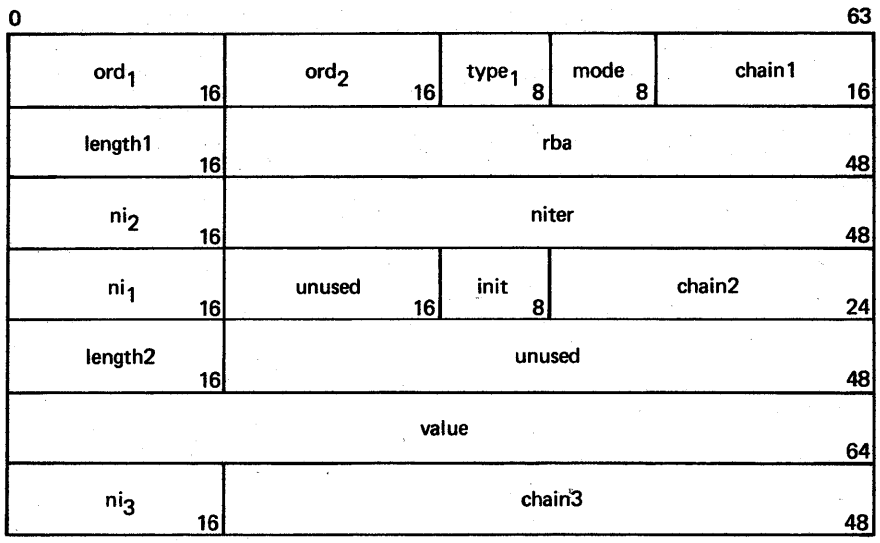
value	Value vector part of the vector to be transmitted; contains a full-word of values (type 5), or a left-justified half-word of values (type 6).
length2	Length of the control vector specified in the string field.
string	A left-justified bit control vector (that is, an order vector).

Figure 9-7. Data Item Format 2 (Sheet 2 of 2)



ord_1	Pseudo address vector table ordinal of static space to be initialized.
ord_2	Pseudo address vector table ordinal of space relative to which relocation is to be performed (relocation base).
type	Type of data item that follows: 7 Full-word index list 8 Half-word index list (not defined if the mode field is 1)
mode	Mode flag: 0 Value to destination 1 Value plus relocation base to destination 2 Destination plus relocation base to destination The meaning of this field is the same as that of the mode field in figure 9-6.
chain	Full-word count to the next data item descriptor in the table.
length	Number of values in the item.
rba	Relative bit address of the location to which the indexed elements of the vector are to be transmitted.
value	A vector; contains a full-word of values (type 7), or a left-justified half-word of values (type 8).
nbi	Number of bits per index.
ni	Number of indexes.
string	A bit string of ni indexes. Each index is nbi bits long and contains a full-word count (for type 7), or a half-word count (for type 8).

Figure 9-8. Data Item Format 3



ord_1 Pseudo address vector table ordinal relative to the data area to be initialized.

ord_2 Pseudo address vector table ordinal of space relative to which relocation is to be performed (relocation base).

$type_1$ D (nested list).

mode Mode flag:

0 Value to destination
1 Value plus relocation base to destination
2 Destination plus relocation base to destination

The meaning of this field is the same as that of the mode field in figure 9-6.

chain1 Full-word count to the next data item in the nested list.

length1 Number of nested item types that follow.

rba Relative bit address of the vector.

ni_2 Nested iteration start item.

niter Number of times the data item and items associated with this iteration start item are to be repeated.

ni_1 Nested data item identifier.

init Any initialization data type. If there is more than one data item in an iteration, types cannot be mixed.

chain2 Length of the data item in number of words.

length2 Half-word vector length.

value A left-justified half-word to be stored in consecutive half-word locations starting at the relative bit address in the rba field.

ni_3 Nested iteration end item.

chain3 Nested item designator:

0 No nested item types follow
1 More nested item types follow

Figure 9-9. Type D Data Item Format

INTERPRETIVE RELOCATION INITIALIZATION TABLE

The interpretive relocation initialization table consists of the two-word loader table header (figure 9-1), followed immediately by one or more relocation items, one word per item. Item formats are similar to data initialization table formats but do not contain values. The name of the table is INT RELO.

TRANSFER SYMBOL TABLE

The transfer symbol table consists of the two-word loader table header (figure 9-1), followed immediately by one word containing the transfer symbol. The table name is XFER SYM. The transfer symbol is the symbolic name of the entry point to which control is to be transferred at the start of execution; the name is left-justified with blank fill.

DEBUG SYMBOL TABLE

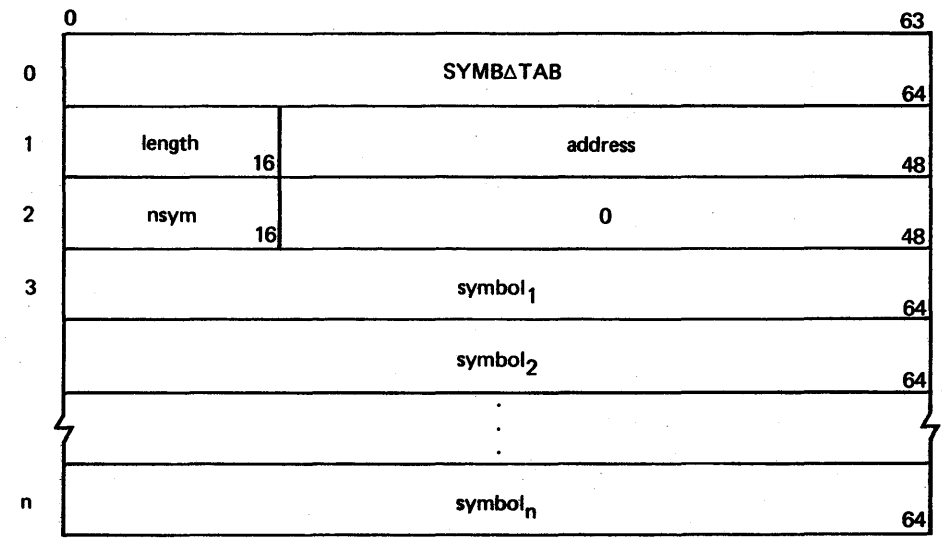
The debug symbol table, which contains the ASCII representation of symbols that appear in a program, allows a symbol to be referenced by name rather than by address. This table appears in the error processing information area if the compiler or assembler used is capable of generating the table, and if the appropriate option is selected and used during compilation or assembly. The format of the table is shown in figure 9-10. The length field in word 2 is the total length of the debug symbol table and the symbol definition table.

SYMBOL DEFINITION TABLE

The symbol definition table is an extension to the debug symbol table. It provides further definition to the debugging symbols, including the type of symbol, address, and mode. The table consists of the two-word loader table header (figure 9-1), followed immediately by one or more two-word entries in the format shown in figure 9-11. The table name is SYMBΔDEF.

PSEUDO ADDRESS VECTOR TABLE

The table pointed to by the ordinal in the symbol definition table is the pseudo address vector table of the data base or common block. The table consists of the two-word loader table header (figure 9-1), followed immediately by two words giving a code address and data base address, and one or more two-word entries in any of the formats shown in figure 9-12. The table name is $\Delta\Delta$PAV$\Delta\Delta$.



nsym — Number of symbols in this table.

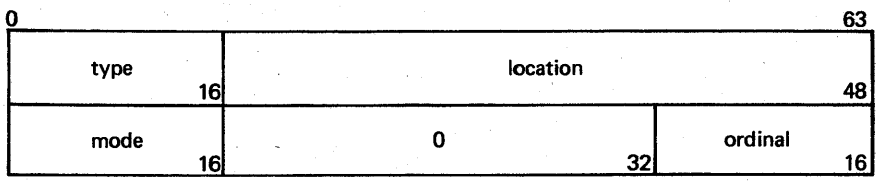
symbol$_i$ — A symbol, which can be any of the following:

Variable or array name, in ASCII; must be left-justified with blank fill.

Statement line number, in ASCII; must be a hexadecimal value, right-justified with ASCII zero fill.

Statement label, in ASCII. Labels that are symbolic names are stored left-justified with blank fill; labels that are statement numbers are stored right-justified with ASCII zero fill.

Figure 9-10. Debug Symbol Table Format



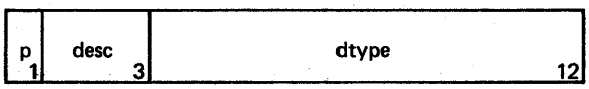
type — Symbol type:

- 0 Unknown
- 1 Half-word register variable name
- 2 Full-word register variable name
- 3 Variable or array name
- 4 Line number
- 5 Label

location — Location field for the symbol type:

- 1 Half-word address within the register file; because half-word values can be stored in full-word registers, the location can range up to hexadecimal 1FF
- 2 Full-word register number
- 3 Bit address relative to the start of the data base
- 4 Bit address relative to the start of the code base
- 5 Bit address relative to the start of the code base

mode — Symbol mode, consisting of three parts as shown below; in the case of a descriptor, P and Dtype describe the contents of the reference vector:



p — Precision base indicator:

- 0 Precision base is 32-bit, or irrelevant
- 1 Precision base if 64-bit

desc — Descriptor indicator:

- 0 Not a descriptor
- 1 Vector descriptor
- 2 Vector descriptor array
- 4 Sparse vector descriptor
- 5 Sparse vector descriptor array

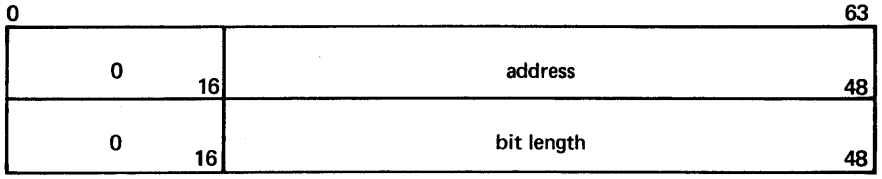
dtype — Type of the referenced vector:

- 0 Unknown
- 1 Logical
- 2 Integer
- 3 Real
- 4 Complex
- 5 Double precision
- 6 Character
- 7 Bit

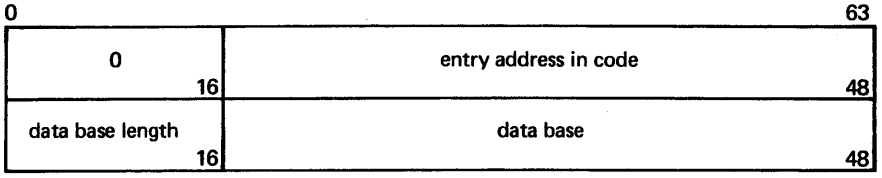
ordinal — Ordinal of the pseudo address vector table of the data base or common block (described under Pseudo Address Vector Table).

Figure 9-11. Symbol Definition Table Entry Format

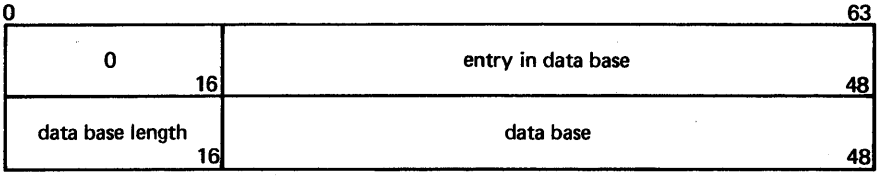
For common:



For an external symbol, referencing the entry point in code:



For an external symbol, referencing the entry point in data:



For an external symbol, referencing a constant entry point :

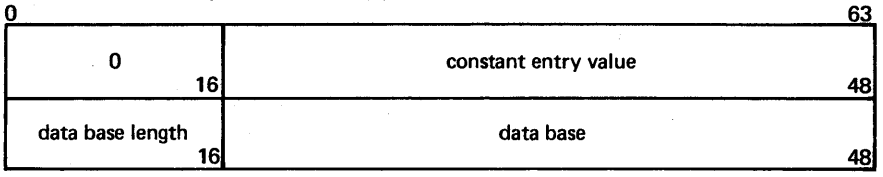


Figure 9-12. Pseudo Address Vector Table Entry Formats

The ASCII character set is shown in table A-1. Aids for hexadecimal-to-octal and hexadecimal-to-decimal conversion are given in tables A-2 and A-3.

TABLE A-1. AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

	b8 b7 b6 b5	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b4 b3 b2 b1	COL / ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	SP no-punch SP 40	0 0 0 F0	@ 8-4 @ 7C	P 11-7 P D7	8-1 79	p 12-11-7 p 97	11-0-9-8-1 DS 20	12-11-0-9-8-1 30	12-0-9-1 41	12-11-9-8 58	12-11-0-9-6 76	12-11-8-7 9F	12-11-0-8 B8	12-11-9-8-4 DC
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	! 12-8-7 | 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8	a 12-0-1 a 81	q 12-11-8 q 98	0-9-1 SOS 21	9-1 31	12-0-9-2 42	11-8-1 59	12-11-0-9-7 77	11-0-8-1 A0	12-11-0-9 B9	12-11-9-8-5 DD
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	" 8-7 " 7F	2 2 2 F2	B 12-2 B C2	R 11-9 R D9	b 12-0-2 b 82	r 12-11-9 r 99	0-9-2 FS 22	11-9-8-2 CC 1A	12-0-9-3 43	11-0-9-2 62	12-11-0-9-8 78	11-0-8-2 AA	12-11-0-8-2 BA	12-11-9-8-6 DE
0 0 1 1	3	ETX 12-9-3 ETX 03	DC3 11-9-3 TM 13	# 8-3 # 7B	3 3 3 F3	C 12-3 C C3	S 0-2 S E2	c 12-0-3 c 83	s 11-0-2 s A2	0-9-3 23	9-3 33	12-0-9-4 44	11-0-9-3 63	12-0-8-1 80	11-0-8-3 AB	12-11-0-8-3 BB	12-11-9-8-7 DF
0 1 0 0	4	EOT 9-7 EOT 37	DC4 9-8-4 DC4 3C	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3	d 12-0-4 d 84	t 11-0-3 t A3	0-9-4 BYP 24	9-4 PN 34	12-0-9-5 45	11-0-9-4 64	12-0-8-2 8A	11-0-8-4 AC	12-11-0-8-4 BC	11-0-9-8-2 EA
0 1 0 1	5	ENQ 0-9-8-5 ENQ 2D	NAK 9-8-5 NAK 3D	% 0-8-4 % 6C	5 5 5 F5	E 12-5 E C5	U 0-4 U E4	e 12-0-5 e 85	u 11-0-4 u A4	11-9-5 NL 15	9-5 RS 35	12-0-9-6 46	11-0-9-5 65	12-0-8-3 8B	11-0-8-5 AD	12-11-0-8-5 BD	11-0-9-8-3 EB
0 1 1 0	6	ACK 0-9-8-6 ACK 2E	SYN 9-2 SYN 32	& 12 & 50	6 6 6 F6	F 12-6 F C6	V 0-5 V E5	f 12-0-6 f 86	v 11-0-5 v A5	12-9-6 LC 06	9-6 UC 36	12-0-9-7 47	11-0-9-6 66	12-0-8-4 8C	11-0-8-6 AE	12-11-0-8-6 BE	11-0-9-8-4 EC
0 1 1 1	7	BEL 0-9-8-7 BEL 2F	ETB 0-9-6 ETB 26	' 8-5 ' 7D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6	g 12-0-7 g 87	w 11-0-6 w A6	11-9-7 IL 17	12-9-8 GE 08	12-0-9-8 48	11-0-9-7 67	12-0-8-5 8D	11-0-8-7 AF	12-11-0-8-7 BF	11-0-9-8-5 ED
1 0 0 0	8	BS 11-9-6 BS 16	CAN 11-9-8 CAN 18	(12-8-5 (4D	8 8 8 F8	H 12-8 H C8	X 0-7 X E7	h 12-0-8 h 88	x 11-0-7 x A7	0-9-8 28	9-8 38	12-8-1 49	11-0-9-8 68	12-0-8-6 8E	12-11-0-8-1 B0	12-0-9-8-2 CA	11-0-9-8-6 EE
1 0 0 1	9	HT 12-9-5 HT 05	EM 11-9-8-1 EM 19	) 11-8-5) 5D	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8	i 12-0-9 i 89	y 11-0-8 y A8	0-9-8-1 29	9-8-1 39	12-11-9-1 51	0-8-1 69	12-0-8-7 8F	12-11-0-1 B1	12-0-9-8-3 CB	11-0-9-8-7 EF
1 0 1 0	10 (A)	LF 0-9-5 LF 25	SUB 9-8-7 SUB 3F	* 11-8-4 * 5C	: 8-2 : 7A	J 11-1 J D1	Z 0-9 Z E9	j 12-11-1 j 91	z 11-0-9 z A9	0-9-8-2 SM 2A	9-8-2 3A	12-11-9-2 52	12-11-0 70	12-11-8-1 90	12-11-0-2 B2	12-0-9-8-4 CC	12-11-0-9-8-2 (LVM) FA
1 0 1 1	11 (B)	VT 12-9-8-3 VT 0B	ESC 0-9-7 ESC 27	+ 12-8-6 + 4E	; 11-8-6 ; 5E	K 11-2 K D2	[12-8-2 ¢ 4A	k 12-11-2 k 92	{ 12-0 { C0	0-9-8-3 CU2 2B	9-8-3 CU3 3B	12-11-9-3 53	12-11-0-9-1 71	12-11-8-2 9A	12-11-0-3 B3	12-0-9-8-5 CD	12-11-0-9-8-3 FB
1 1 0 0	12 (C)	FF 12-9-8-4 FF 0C	FS 11-9-8-4 IFS 1C	, 0-8-3 , 6B	< 12-8-4 < 4C	L 11-3 L D3	\ 0-8-2 \ E0	l 12-11-3 l 93	| 12-11 | 6A	0-9-8-4 2C	12-9-4 PF 04	12-11-9-4 54	12-11-0-9-2 72	12-11-8-3 9B	12-11-0-4 B4	12-0-9-8-6 CE	12-11-0-9-8-4 FC
1 1 0 1	13 (D)	CR 12-9-8-5 CR 0D	GS 11-9-8-5 IGS 1D	– 11 – 60	= 8-6 = 7E	M 11-4 M D4	] 11-8-2 ! 5A	m 12-11-4 m 94	} 11-0 } D0	12-9-8-1 RLF 09	11-9-4 RES 14	12-11-9-5 55	12-11-0-9-3 73	12-11-8-4 9C	12-11-0-5 B5	12-0-9-8-7 CF	12-11-0-9-8-5 FD
1 1 1 0	14 (E)	SO 12-9-8-6 SO 0E	RS 11-9-8-6 IRS 1E	. 12-8-3 . 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ¬ 5F	n 12-11-5 n 95	~ 11-0-1 ~ A1	12-9-8-2 SMM 0A	9-8-6 3E	12-11-9-6 56	12-11-0-9-4 74	12-11-8-5 9D	12-11-0-6 B6	12-11-9-8-2 DA	12-11-0-9-8-6 FE
1 1 1 1	15 (F)	SI 12-9-8-7 SI 0F	US 11-9-8-7 IUS 1F	/ 0-1 / 61	? 0-8-7 ? 6F	O 11-6 O D6	_ 0-8-5 _ 6D	o 12-11-6 o 96	DEL 12-9-7 DEL 07	11-9-8-3 CU1 1B	11-0-9-1 E1	12-11-9-7 57	12-11-0-9-5 75	12-11-8-6 9E	12-11-0-7 B7	12-11-9-8-3 DB	EO 12-11-0-9-8-7 FF

64-Character ASCII Subset

96-Character ASCII Subset

LEGEND

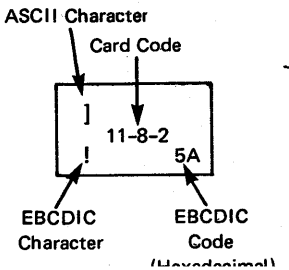


TABLE A-2. HEXADECIMAL-OCTAL CONVERSION TABLE

Second Hexadecimal Digit	First Hexadecimal Digit 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	000	020	040	060	100	120	140	160	200	220	240	260	300	320	340	360
1	001	021	041	061	101	121	141	161	201	221	241	261	301	321	341	361
2	002	022	042	062	102	122	142	162	202	222	242	262	302	322	342	362
3	003	023	043	063	103	123	143	163	203	223	243	263	303	323	343	363
4	004	024	044	064	104	124	144	164	204	224	244	264	304	324	344	364
5	005	025	045	065	105	125	145	165	205	225	245	265	305	325	345	365
6	006	026	046	066	106	126	146	166	206	226	246	266	306	326	346	366
7	007	027	047	067	107	127	147	167	207	227	247	267	307	327	347	367
8	010	030	050	070	110	130	150	170	210	230	250	270	310	330	350	370
9	011	031	051	071	111	131	151	171	211	231	251	271	311	331	351	371
A	012	032	052	072	112	132	152	172	212	232	252	272	312	332	352	372
B	013	033	053	073	113	133	153	173	213	233	253	273	313	333	353	373
C	014	034	054	074	114	134	154	174	214	234	254	274	314	334	354	374
D	015	035	055	075	115	135	155	175	215	235	255	275	315	335	355	375
E	016	036	056	076	116	136	156	176	216	236	256	276	316	336	356	376
F	017	037	057	077	117	137	157	177	217	237	257	277	317	337	357	377
Octal	000 – 037		040 – 077		100 – 137		140 – 177		200 – 237		240 – 277		300 – 337		340 – 377	

TABLE A-3. HEXADECIMAL-DECIMAL CONVERSION TABLE

		Exponent for Base 16					
		5	4	3	2	1	0
Hexadecimal Number	0	0	0	0	0	0	0
	1	1048576	65536	4096	256	16	1
	2	2097152	131072	8192	512	32	2
	3	3145728	196608	12288	768	48	3
	4	4194304	262144	16384	1024	64	4
	5	5242880	327680	20480	1280	80	5
	6	6291456	393216	24576	1536	96	6
	7	7340032	458752	28672	1792	112	7
	8	8388608	524288	32768	2048	128	8
	9	9437184	589824	36864	2304	144	9
	A	10485760	655360	40960	2560	160	10
	B	11534336	720896	45056	2816	176	11
	C	12582912	786432	49152	3072	192	12
	D	13631488	851968	53248	3328	208	13
	E	14680064	917504	57344	3584	224	14
	F	15728640	983040	61440	3840	240	15

	i
j	m

$j_{16} \times 16^{i} = m_{10}$

To find $E_{16} \times 16^{3}$; look at row E, column 3 and find 57344

This appendix describes the meanings of the system error and dead codes.

SYSTEM ERROR CODES

The error codes listed in table B-1 are returned in word #8B of the minus page. The errors signified by the codes terminate the task that generated the error. The KERNEL, RESTART, and RECOVERY tasks are part of the resident operating system; the AOK and SCAT tasks are part of the virtual system.

SYSTEM DEAD CODES

A dead code is the product of an error detected in a part of the operating system and renders the system inoperative. The codes, listed in table B-2, are grouped according to the part of the system affected. All dead codes are in hexadecimal.

TABLE B-1. SYSTEM ERROR CODES

Error Code	Significance	Issued by
5	Illegal instruction.	KERNEL
6	No message pointer follows the exit force instruction.	KERNEL
7	Illegal request.	KERNEL
8	Memory parity error.	RESTART
9	Fatal system error.	RECOVERY
A	A user other than the virtual system made a C500, C501, C502, or C503 call with a file segment table (FST) ordinal of zero.	Resident system
B	A user other than the virtual system issued a C504 call.	Resident system
B-xx	File is already extended to maximum. xx is an input/output connector number.	EXTEND
C-xx	Attempted to read past the physical space. xx is an input/output connector number.	REXTEND
D-xx	No segments left for extension. xx is an input/output connector number.	EXTEND
E-xx	No space left on the disk for extension. xx is an input/output connector number.	EXTEND
24	Large page limit exceeded.	PAGER
28	Write violation occurred for a file that did not have write access or for a write-temporary file.	PAGER
29	User fault for a page in the virtual system address range.	PAGER
2A	Attempted to make a drop file map entry when the drop file does not exist, or a drop file map overflow occurred during an attempt to make a drop file map entry.	PAGER
2C	User is faulting for library space.	PAGER
2D	Drop file space overflow.	PAGER
2E	Page was not mapped because the drop file map is full.	PAGER
2F	Drop file overflow was caused by a call to the virtual system.	PAGER
30	No system resource charges are available for this task.	PAGER

TABLE B-1. SYSTEM ERROR CODES (Contd)

Error Code	Significance	Issued by
31	Input/output error response was received by WRPLY.	PAGER
40 through 4F	Reserved for BATCHPRO.	Virtual system
50	File to be extended was not found.	EXTEND
51	File segment table (FST) is full.	EXTEND
FF	Read or write failed due to a disk error.	PAGER
209	No source file exists.	AOK
210	No drop file exists.	AOK
212	No Alpha pointer exists.	SCAT
213	Alpha pointer is out-of-bounds.	SCAT
215	No error exit address exists.	SCAT

TABLE B-2. SYSTEM DEAD CODES

Error Code	Significance	Issued by
	RESINIT Dead Codes	
1	Checksum error occurred in a station logon.	RESINIT
2	First message received from the station was not a logon.	RESINIT
3	Checksum error occurred on a station initialization response.	RESINIT
4	Unused.	
5	Problem with virtual system source or paging file device.	RESINIT
6	Page frame in the system area was bad.	RESINIT
7	Paging File is too small.	RESINIT
	KERNEL Dead Codes	
11	Job is illegal; the p-counter address is not in core - virtual system only.	KERNEL
12	Illegal instruction in the virtual system.	KERNEL
13	Illegal instruction in the resident system.	KERNEL
14	Invalid message indicator from the virtual system.	KERNEL
15	Illegal virtual system call from PAGER.	KERNEL
16	Illegal function code from the virtual system.	KERNEL
17	Message has an invalid station number.	KERNEL
18	Virtual system page zero is missing.	KERNEL
19	User's page zero was missing before the run.	KERNEL
1A	KERNEL cannot find page zero for the user at initialization time.	KERNEL

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
1B	Checksum error on a station request.	KERNEL
1C	Attempt to release a page not reserved by the station.	KERNEL
1D	An I/O request for a segmented file required more than the maximum number of betas.	KERNEL
1E	Checksum error on a station response.	KERNEL
1F	Message page is not in core.	KERNEL
20	No virtual process matches the resume address.	KERNEL
21	Too many parameters in a KERNEL call.	KERNEL
22	Illegal call for a KERNEL function.	KERNEL
23	Duplicate page table entry.	KERNEL
24	Attempt to delete a locked page.	KERNEL
25	List buffer exceeds a page.	KERNEL
26	Missing page table entry.	KERNEL
27	DTQ or STQ queue overflowed.	KERNEL
28	Invalid service station request received.	KERNEL
29	Missing page table entry.	KERNEL
2A	Overflow to the message table.	MESTAB
2B	Unused.	
2C	User's virtual zero page was not found.	KERNEL
2D	User's zero page is not in PAGTAB.	KERNEL
2E	User's zero page is not in VPZTAB (that is, not in PAGTAB).	KERNEL
2F	Unused.	
30	Using a C500, C501, C502, or C503 call, the requested sector was not represented in the file segment table.	KERNEL
	SUPPORT Dead Codes	
31	System task queue overflow.	TASKQ
32	Invalid message in a call to FAULRET.	SUPPORT
33	Missing page table entry.	FAULRET
34	Unused.	
35	Duplicate page table entry detected for a page.	FF01
36	New page entry missing from the page table.	FF01
37	Invalid page table entry for a requested deletion.	FF02
38	Missing page table entry for a requested deletion.	FF02
39	Unused.	
3A	Call spanned more than two pages.	FINDRAN

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
3B	Free page list overflow.	FREEPAGE
3C	Free page stack overflow.	FREEPAGE
3D	Free page list overwrite.	FAULRET
3E . . .	Unused. . . .	
4C	Unused.	
4D	Free boat not available.	BTISSUE
	Virtual System Dead Codes (102-131)	
102	Cannot recognize the USER1 name.	STUSR1
103	Error during call to get input buffer.	F00D/ SEQIO
104	User table entry was not found upon completion of writing user directories to the drum.	UDM
105	Error return on a C307 call.	UDM
106	File management returned an error on the execute line.	TTYMES
107	Unused	
108	MESP and PROG are nonzero in the BYE path.	TTYMES
109	Teletypewriter terminal unit (TTY) is logged in; error and the last bit is set.	TTYMES
10A	Program state of the problem program in the DBMIDL queue is less than #1E.	MIDL
10B	User table in the output chain, not a break with descriptor block number zero.	PPOUT
10C	File name error: zero or a USER1 user number.	SASP
10D	No problem program controller when not attached to the TTY.	FINDMP
10E	Job task table is full.	VINIT or SCH_INIT
10F	DBIN or DBOUT do not point to a valid descriptor block; or, a message from a controllee is already there.	TOCON
110	Error received from file management.	ICNTE/ ICNTECH
111	Input/output call buffers are full.	FINBUF
112	No REGTBL entry was found.	INTRP
113	At least one STCK interrupt should be set.	RFI
114	User not in the user table.	MISMSG/ STOPMG
115	Virtual system tables overlap code.	Virtual system
116	Error on the first read of a user directory entry.	VINIT
117	Error on the first write of a user directory entry.	VINIT

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
118	Error on the second read of a user directory entry.	VINIT
119	Error on verifying a user directory entry.	VINIT
11A	Division is incorrect.	VINIT
11B	TTY buffer page is not in the page table.	TTYMES
11C	Page is not in the page table.	VINIT
11D	Unused.	
11E	Unused.	
11F	Error on writing a public file to disk.	VINIT
120	Error in creating a system drum file.	VINIT
121	Error in writing system pages to the drum file.	VINIT
122	Unused.	
123	Cannot find a page in the list range.	SEQIO
124	Unused.	
125	Bad function code encountered upon completion of sequential input/output (greater than 4).	SQFIN
126	No Alpha pointer in the register parameter.	SEQIO
127	Page was not in the page table after the fault.	SEQIO/ TTYMES
128	No external in the data base.	VINIT
129	Unable to requeue the service station input/output files.	Virtual system
130	A task was put into the problem program output chain but has no output.	PPOUT
131	System processor table is full.	VINIT
132	Unable to switch keys/extend dayfile.	SMTDF
	PAGER Dead Codes	
200	Minus page reference for a nonexistent descriptor block.	PAGER
201	Access interrupt; cause bits are in error (bits 12 and 13 are zero).	PAGER
202	Write violation but the virtual address is not in the bound virtual map.	PAGER
203	Virtual system faulting for an illegal address.	PAGER
204	Zero entry found in the drop file map.	PAGER
205	Unused.	
206	Unused.	
207	Unused.	
208	Unable to allocate a virtual system or special region page.	PAGER
209	Unused.	
20A	Virtual address is in an invalid range.	PAGER

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
20B	Error during storage move to free a large page.	PAGER
20C . . . 210	Unused. . . . Unused.	
211	Bad PAGER request from KERNEL.	PAGER
212	Unused.	
213	Unused.	
214	Virtual system faulting for an address in the user range without a user key.	PAGER
215 . . . 221	Unused. . . . Unused.	
222	Virtual system faulting for a user page with no physical space for the page.	PAGER
223	PAGER is attempting to swap pages of differing sizes.	PAGER
224	Page is missing its entry in the page table.	PAGER
225	Entry in the page table, but the page was not represented in the drop file or in the bound virtual maps.	PAGER
226	Working set marker page table entry for this alternator entry is not in the page table.	PAGER
227	Input/output error and descriptor block number in request are zero. zero.	WRPLY
228	Unused.	
229	Faulting task does not have an entry in the CPU queue.	PAGER
	Virtual System Dead Codes (301-402)	
301	XEQBUF is empty; or error response on a disk read; or NEXB is greater than FMCE.	CREATE
302	Bad NFPP pointer detected in call to RELMES.	HANDLE
303 . . . 305	Unused. . . . Unused.	
306	Error on the delete range from core.	PURGE
307	Error returned from FMAKER during re-creation of the original drop file after an error occurred during a user-generated drop file creation.	CREATE
308	Error returned from ENTRPFI during re-creation of the original drop file after an error occurred during a user-generated drop file creation.	CREATE
309	Error on a disk read.	OPEN
30A	Error response on a translation of a virtual address (F007 call).	OPEN

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
30B	MP3BUF page was not found.	OPEN
30C	Error on deletion of MP3BUF from the page table.	OPEN
30D	Error on deletion of a range from the page table.	MAP
30E	Error in F007.	MAP
30F	Error on deletion of the virtual range from the page table.	MAP
310	Error on writing a page to the disk.	MAP
311	Error on reading a page from the drum.	MAP
312	Overlap detected, or entry extending beyond end of disk detected, while scanning DMAP.	FILOC
313	Anomaly detected in a virtual map.	MAP
314	Overlap detected while compressing DMAP.	FILOC
315	Error on deletion of a range from core.	CLOSE
316	Error on translation of a virtual address.	CLOSE
317	Error on deletion of a range from the page table.	CLOSE
318	Error on reading a page from the drum.	CLOSE
319	Unused.	
31A	Error on writing a page to disk.	CLOSE
31B . . .	Unused. . . .	
329	Unused.	
32A	Error on writing user directory to the drum file.	GIVEF
32B	Error on deleting Pack File Index entry.	GIVEF
32C . . .	Unused. . . .	
33A	Unused.	
33B	Cannot locate job in JOB LIST table.	RESTART
33C	I/O error on checkpoint.	KERNEL
33D	I/O error on RESTART.	KERNEL
33E	System task queue not empty on RESTART.	KERNEL
33F	Unable to start up OPERATOR or unable to store OPERATOR message on RESTART.	RESTART
340	Program is in the wait queue, but is not in the program queue.	LOAD
341	Number of programs in the wait queue does not match the MISCTAB variable NOWAITQ.	LOAD
342	Number of programs in the CPU queue does not match the MISCTAB variable NOCPUQ.	LOAD
343	Program is not in the program queue.	SCHEDULE, ENTER_WQ

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
344	Program is already in the wait queue.	SCHEDULE, ENTER_WQ
345	Program is not in the program queue.	SCHEDULE, DELETE_WQ
346	Subpriority exceeds the maximum when reassigning from the minimum.	SCHEDULE, CREATE_PQ
347	Program is not in the program queue.	SCHEDULE, ENTER_CPUQ
348	Program is in the wait queue and the alternator.	SCHEDULE, ENTER_CPUQ
349	Minus page is not in the page table.	SCHEDULE, ENTER_CPUQ
34A	Program is already in the CPU queue.	SCHEDULE, ENTER_CPUQ
34B	Program is not in the CPU queue.	SCHEDULE, DELETE_CPUQ
34C	Program is not in the program queue.	SCHEDULE, DELETE_CPUQ
34D	Unused.	
34E	Unused.	
34F	Unused.	
350	Interrupt with no REGTBL pointer.	DDUMP
351	Bad previous state.	DDUMP
352	Unused.	
.	.	
.	.	
.	.	
381	Unused.	
382	Unable to delete/unlock physical pages.	UDINIT
383	Q5UD file too small.	UDSRCH
384	Unused.	
385	Cannot find Q5UD file.	UDSRCH
386	Error on C500 read.	UDRDWR

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
387	Error on C501 write.	UDRDWR
388	Cannot find Q5UD file.	UDRDWR
389	Unused.	
390	Unused.	
391	Unused.	
392	Unable to delete/unlock physical pages.	UDMODD
393 . . . 3FF	Unused. . . . Unused.	
400	Error on a service station read.	VINIT
401	Illegal program state found by the loader.	DISCON
402	Input queue manager not found in SPT table.	OPCOM
	Permanent Files Dead Codes (500-592)	
500	Error in C500 during a read of the first page of the pack file index (PFI).	PACKSUP/ UFX
501	Error in F007.	PACKSUP
502	Error in C500 during a read of one page at a time of the PFI.	PACKSUP
503	Error in F007.	MODPFI
504	Error in C500.	MODPFI
505	Failed to find a page for a disk read.	GETFI
506	Failed to read a page of the PFI.	GETFI
507	Failed to release a page of disk read.	GETFI
508	Failure in a C501 write.	MODPFI
509	Error in F002.	MODPFI
50A . . . 52F	Unused. . . . Unused.	
530	Error returned from ENTRPFI.	NEWAF
531	Error returned from ENTR.	NEWAF
532	Error in a C500 call.	ASINIT
533	Error in a C501 call.	ASINIT
534	Error in F007.	ASINIT
535	Error in F007 call (find page for accounting table).	WRTAF
536	Error in C501 call (write accounting table).	WRTAF

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
537	Error in C501 call (update header/terminate file).	WRTAF
538	Unused.	
539	Error in C501 call (initialize or update header).	WRTAF
53A . . . 53F	Unused. . . . Unused.	
540	Cannot find *AF file in pack file index.	WRTAF
541	The variable RTYPE is out of range.	ACCTG
542	Next block in *AF file is outside of the file.	ASINIT
543 . . . 55F	Unused. . . . Unused.	
560	Error in F007 or an F007 call.	GETPFI
561	Error in a C500 call.	GETPFI
562	Error in a C500 call.	GETPFI
563 . . . 57F	Unused. . . . Unused.	
580	Error in F007.	BADSEC
582 . . . 58F	Unused. . . . Unused.	
590	Error in C500 during the read of a bad sector map. Look at DISF to see which pack it is. Find the bad sector PFI entry to locate the bad block.	PACKSUP
591	The count of bad sectors in word 0 of the bad sector file is greater than 255. Look at DISF to see which pack it is. Find the bad sector PFI entry to locate the bad sector block. Check for garbage in the bad sector block. Call a customer engineer to redo the map.	PACKSUP
592	Word 12 of the label contains a BASEC file name, but there is no entry in the PFI. Look at DISF to see which pack it is. Check the label and the PFI for that pack. Relabel if necessary.	PACKSUP
	Pool File Manager Dead Codes	
600	Could not find input/output in the page table.	FINDPL
601	Could not open the pool list (PLIST) or pool access directory (PAD) files; or, could not close the PLIST or PAD file.	FINDPL
602	Could not read the pool list (PLIST) file.	FINDPL

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
603	Could not delete input/output pages from the page table.	CLPLFI
604	Could not write the pool list (PLIST) file.	WRITPL
605	Could not read the pool access directory (PAD) file.	READPD
606	Could not write the pool access directory (PAD) file.	WRITPD
	Permanent Files Dead Codes (700-801)	
700	ENTRPFI returned an error during the write of a history file PFI entry.	HISTINIT
701 . . . 7FF	Unused. . . . Unused.	
802	Local file has a PFI entry.	GATHRAFI
803	Error in deallocating a segment.	GATHRAFI
805	Destroy error	GATHRAFI
	Recovery Dead Codes	
900	Could not find loader tables for the recovery module. Module must be reloaded in the maintenance control unit.	KERNEL
901 . . . 905	Unused. . . . Unused.	
906	System initialization table (SIT) was improperly set up by the maintenance control unit during parity error recovery.	RESTART
907	Parity error recovery called for a non-CPU parity error.	RESTART
908	Parity error occurred in a locked virtual system page.	RESTART
909	Page address of the parity error is not in core.	RESTART
90A . . . 90F	Unused. . . . Unused.	
910	Boat linkage was bad during recovery from dead.	RECOVERY
911	Descriptor block for the currently executing task does not have an alternator slot.	RESTART
912	Unused.	
913	Unused.	
914	Unused.	
915	Minus page of the task being recovered is not in core.	RECOVERY
916 . . . 920	Unused. . . . Unused.	

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
921	Excessive number of batch processors (BATCHPRO) were aborted during recovery.	RECOVERY
922	Unused.	
923	Unused.	
924	Unused.	
925	Zero file segment table (FST) entry was found during recovery.	RECOVERY
926	Cannot find the disk address in the file segment table (FST) for this file.	RECOVERY
927 . . .	Unused. . . .	
92F	Unused.	
930	Station timed out; not responding to the message.	RSUPPORT
931	Bad response from the station during recovery.	RSUPPORT
932	Unused.	
933	Unused.	
934	Unused.	
935	Page table entry was not found.	RSUPPORT
936 . . .	Unused . . .	
93F	Unused.	
940	Read error while trying to read a file from the service station.	VRECOVRY
941	Open error in a service station file.	VRECOVRY
942	Input buffer did not exist during read of a service station file.	VRECOVRY
943	Close error on a service station file.	VRECOVRY
944	Destroy error on a service station file.	VRECOVRY
945	File index table was full on a call to GETFI.	VRECOVRY
946	Cannot find the drop file for a recoverable task.	VRECOVRY
947	File type is bad for this drop file.	VRECOVRY
948	Input/output connector for the source or drop file is missing.	VRECOVRY
949	SEARCH could not find the public or attached permanent file name.	VRECOVRY
94A	Excessive number of batch processors (BATCHPRO) were aborted during recovery.	VRECOVRY
	Temporary Dead Codes (A55)	
A55	Two segment allocation errors.	ENTR
	Permanent File Dead Codes (B01-B41)	
B01	Bad response on reading the first sector of the PFI (function C500).	Virtual system

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
B02	Error response on getting a buffer for reading or writing the pack file index (function F00D).	Virtual system
B03	Error response on the list of buffer entries (function F004).	Virtual system
B04	Error response on reading the pack file index (function C502).	Virtual system
B05	Error response on writing the pack file index (function C503).	Virtual system
B06	Error response on releasing the buffer (function F003).	Virtual system
B07	Error returned on a list page table entry for the label buffer (function F007).	Virtual system
B08	Error returned on a rewrite of the pack label (function C501).	Virtual system
B09 . . . B10	Unused. . . . Unused.	
Bxx	Unable to find a good label; the logical unit number (xx) is #11 to #20.	Virtual system
B21 . . . B30	Unused. . . . Unused.	
Byy	Too many bad sector entries; the logical unit number (yy) is #31 to #40.	Virtual system
B41 . . . B4F	Unused. . . . Unused.	
B50	F00D error returned during PFI processing.	GETBUF
B51	F004 error returned.	GETBUF
B52 . . . B5B	Unused. . . . Unused.	
B5C	PFI buffer overflow.	PROCPFI
B5D	PFI buffer overflow.	PROCPFI
B5E	PFI buffer overflow.	PROCPFI
B5F . . . B62	Unused. . . . Unused.	.

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
B63	PFI read request error.	PACKSUP
B64	Unused.	
B65	Reserved.	
B66	Too many bad sectors for DMAP sort.	PACKSUP
B67	File create error for IFI file.	PACKSUP
B68	File create error for PLIST file.	PACKSUP
B69	File create error for PAD file.	PACKSUP
B6A	F003 error returned.	PACKSUP
B6B . . . B7F	Unused. . . . Unused.	
B80	No IFI file exists at build time.	BUILDMFI
B81	Unused.	
B82	IFI file overflow.	BUILDMFI
B83 . . . B8F	Unused. . . . Unused.	
B90	Block to deallocate not in use in IFI file.	DEALLOCIFI
B91	Could not allocate IFI file space.	ALLOCIFI
B92	Unused.	
B93	Unused.	
B94	Bad sector count parameter for read/write.	WRITEIFI
B95	Read error returned (C500).	WRITEIFI
B96	Write error returned (C501).	WRITEIFI
	Virtual System I/O Dead Codes (D01-D02)	
D01	Zip code associated with a disk file was not found in the table of zip codes.	DDUMP
D02	The change of the state code of a DB with outstanding I/O was caused by an event other than a break ( ).	DDUMP
D03	No name in mapped IOC.	DDUMP
	File Extension Dead Codes (E02-E04)	
E02	Error in updating the pack file index during extension.	EXTEND
E03	Two segment allocation errors during extension.	EXTEND
E04	Discrepancy between the file segment table entry and pack file index segment information.	EXTEND
E05	Unused.	

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
	Virtual System I/O Dead Codes (F01-F09)	
F01	DB being processed is not in a dump state and has no alternator.	WRITER
F02	The table of identifiers is full and writer has been called for a new identifier.	WRITER
F03	The table associating identifiers and I/O requests is full.	WRITER
F04	No identifier can be found for a completed I/O request.	WRITER
F05	WRITFIN was queued but no completed I/O request was found.	WRITFIN
F06	Error from I/O request.	WRITER
F07	Error from post I/O request.	WRITFIN
F08	Page locked for C503 call was found already unlocked upon request completion.	WRITFIN
F09	Buffer page for C503 call in use, or buffer page was not found for C503 call.	QE_FBUF
	Permanent File Dead Codes (4200-4704)	
4200	Unused.	
4201	Unrecognizable pack type was indicated in the pack label (841, for instance).	PACKSUP
4202	DELPFI error.	PACKSUP
4203	MAKEPFI error.	PACKSUP
4204	DDUMP cannot find batch job in input queue.	DDUMP
4205 . . . 4220	Unused. . . . Unused.	
4221	Error in setting up DMAP indicated overlapping files on the pack. Hand-eliminate one file.	PACKSUP
4222	Bad sector overlaps with file.	PACKSUP
4223	Overflow of DMAP table or new entry.	PACKSUP
4224	Unused.	
4225	Duplicate pack labels detected. Relabel one pack and try again.	PACKSUP
4226 . . . 4230	Unused. . . . Unused.	
4231	PFI entry in the pack file index is not properly labeled; hand-change the label.	PACKSUP
4232	Too many file fragments for the dimensions of the sorting array in PACKSUP. Increase the dimensioned size of the array and recompile.	PACKSUP
4233 . . . 4701	Unused. . . . Unused.	

TABLE B-2. SYSTEM DEAD CODES (Contd)

Error Code	Significance	Issued by
	Permanent File Dead Codes (4200-4704) (Contd)	
4702	Nonprivileged user issued a GET PACK LABEL AND PFI (function 11) message.	Virtual system
4703	Disk was not up when a GET PACK LABEL AND PFI message was issued.	Virtual system
4704	Cannot find the pack identified in a GET PACK LABEL AND PFI message.	Virtual system
	CYBER 200 Resident/CYBER 200 Link Dead Codes (5001-5028)	
5001 through 5028	Reserved.	
	CYBER 200 Virtual/CYBER 200 Link Dead Codes	
5100, 5101	Reserved.	
	CYBER 200 Resident/CYBER 200 Link Dead Codes (5121-5128)	
5121 through 5128	Reserved.	
	Performance Measurement Dead Code (7001-7003)	
7001	Performance measurement virtual request stack overflow.	KERNEL
7002	Performance measurement resident request stack overflow.	KERNEL
7003	Performance measurement function code error.	KERNEL
	Temporary Dead Codes (AAAA-CACA)	
AAAA	Trap for dead code 4221 caused by duplicate accounting files.	WRTAF
BEEF	Alternator call-block bit (SCBIT) has been cleared at the completion of a system call. Bit should still be set.	SCAT
CACA	Virtual system has received an input message from terminal 0. Usually means service station trouble.	TTYMES
	Close File Dead Codes (CF01-CF05)	
CF01	File not found in the file index.	CLOSFIN
CF02	System call bit not set in the alternator.	CLOSFIN
CF03	Error in modifying the permanent file index.	CLOSFIN
CF04	Error in a privileged pool detach.	CLOSFIN
CF05	Error in an F00F call.	CLOSFIN
	Temporary Dead Codes (FACE-FECE)	
FACE	Operator-induced dead caused by resident system variable patch=0.	KERNEL
FD01		FINDMP
FECE	Message buffer MFLINE is about to be overemptied.	MISOUT

Access

A parameter that specifies the read or write access desired for a file at the time the file is opened or created. The system grants access only if the lockout field of the file index table allows such access.

Access Station

A CYBER computer system running under control of the NOS operating system that is used to enter jobs into the CYBER 200 system and to control peripheral devices.

Account Identifier

One to eight characters indicating who is to be charged for system resource usage attributable to a user number.

Batch Job

A task that is executed as a controllee of the batch processor.

Batch Processor

A system utility that processes batch jobs. Control statements that are file names cause the files named to be executed as controllees of the batch processor.

Bound Implicit Map

Part of the minus page of an executing file that relates virtual addresses with physical mass storage addresses. An entry is made in a bound implicit map every time a file is opened for implicit input/output.

Byte

A sequence of eight bits that is a subdivision of a word and represents a single character.

Checkpoint

A system feature that captures a task and any of its controllees at some point into execution such that the task can be restarted from that point. Checkpoint is called through a FORTRAN program by the file name CHKPNT.

Controllee

A task called into execution by a controller.

Controllee Chain

A linked series of tasks that results when one task brings another task into execution. That task can, in turn, initiate another task. As many as five levels of tasks can be involved. The first level is level 1; the last is level 5.

Controllee File

See Virtual Code File.

Controller

A relative term that indicates that a member of a controllee chain has a controllee task attached. A controller might be a controllee of another task. Two controllers that themselves have no controllers (that is, two level-1 tasks) are the batch processor and virtual system interactive processor.

CPU

Central processing unit, the computational facility of the CYBER 200 system.

CYBER 200 Link Station

A CYBER computer system running under control of the NOS/BE operating system that is used to enter jobs into the CYBER 200 system and to control peripheral devices.

Data Base

The constants and variables used by a routine, not including entities declared to be in common.

Dayfile

A file produced by the batch processor for a batch job which gives a history of the job. Information on the file includes the time various control statements began execution and any error or status information produced by system utilities. The dayfile is printed as part of job output.

Descriptor Block Number

A unique number associated with the program until it terminates. This number is the key link between the operating system and an executing program. (If the controller is interactive, the field is set to #FF, and 0 for BATCHPRO).

Drop File

A file management category of a file created by the system for modified pages of an executing task, free space, and write-temporary files.

Drop file names are formed by the system shifting the controllee file name right two characters and prefixing it with digits that identify the suffix (digits 1 through 4 corresponding to suffixes A through D) and the level 1 through 5 in a controllee chain.

Drop File Map

Part of the minus page of an executing file that relates virtual addresses with physical mass storage addresses. An entry is made in the drop file map every time a free-space attachment is made by the executing code.

Explicit Input/Output

A means of accessing a mass storage or tape file in which data is buffered under program control. Contrast with Implicit Input/Output.

Family File

A set of files with names that begin with Pnn, which is printed at job termination when a file with a name of PXX is added to the family.

File

A collection of data that can be accessed by file name. In the absence of an adjective such as card or tape, all references to files in this manual imply mass storage files.

File Index Table

A system table that holds all information relating to file characteristics.

File Type

A category that defines file structure from a system standpoint. File types are: physical data, virtual data, and virtual code.

Implicit Input/Output

A means of accessing a mass storage file in which the system brings a page of the file into main memory in response to a reference on that page. Contrast with Explicit Input/Output.

Input/Output Connector (IOC)

An entry in a minus page that links a file with a task for input/output purposes.

Invisible Package

A hardware convention that contains the address and control information for the corresponding job.

Job

See Batch Job.

Labeled Tape

A magnetic tape with labels conforming to American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange.

Large Page

128 small pages; 65 526 contiguous words of 64 bits.

Level

Depending on context, can refer to the security level of a file, the level of a file in the controllee chain, or the level of a routine involved in an interrupt. For the first and second meanings, see Security Level and Controllee Chain.

With respect to interrupt processing, level 0 refers to the currently executing routine; and level 1, when an interrupt has been requested, refers to the interrupting routine until that routine starts executing.

Library

A file of modules, in a format produced by the system utility OLE, that can be used to satisfy external references during loading.

Local File

A private file that is destroyed by the system after termination of the batch job or interactive terminal session that creates it.

Lockout

A field in the file index table that defines whether the file can be read or written. The field is set at the time a file is created. It defines access in terms of what is not allowed. Also see Access.

Main Memory

Memory associated with the central processing unit from which instructions can be executed. Also called MCS.

Management

A category that defines what if any special processing a file is to receive after task execution. Management categories are mass storage, output, scratch, write-temporary, magnetic tape, and system-generated or user-generated drop files.

Map

See Bound Implicit Map or Drop File Map; also MAP system message.

Mass Storage File

A file management category that indicates no special processing after task termination. In a general sense, mass storage indicates disk-resident files, as opposed to magnetic tape files.

Message

See System Message.

Minus Page

The first page of a virtual file used by the system to hold items such as the invisible package, input/output connector information, and maps of defined virtual space. Drop files can also have a second minus page containing overflow input/output connector and map information.

Nonprivileged

A status that allows access to files owned by the same user number under which the task is running, to public files, and to authorized pool files. Contrast with privileged.

Object Code File

A file generated by compilation or assembly of a source language program that can be used by the loader to produce an executable file.

OLE

System utility that creates and modifies a file in library format or modmerge file format.

Output File

A file management category that indicates a file is destined for print or punch equipment.

Also, a generic term for a file being written, as opposed to an input file being read.

Ownership

A category for each file that determines what nonprivileged tasks can access a mass storage file. Ownership categories are public, private, and pool.

File names can be duplicated in different categories for a single user number. A private file can have the same name as a public file, and two user numbers can each be associated with a private file with the same name.

Pack File Index (PFI)

A table of 16-word File Index Table entries which exists on each pack to control the files located on that pack.

Page

The unit by which main memory is managed; a block of contiguous 64-bit words. Can be a large page or small page.

Page Fault

Reference by virtual address to a page not currently in main memory, causing a program interrupt and paging in.

Page Zero

The second page of a virtual code file which contains register contents when a task is not executing in the CPU.

Paging In

Operation to move a page from auxiliary memory to main memory.

Paging Out

Operation to move a page from main memory to auxiliary memory.

Permanent File

A private file that remains in the system after termination of the batch or interactive session that creates it.

Physical Data File

A file type that indicates a file containing nonexecutable data only.

Physical Memory Address

Address of a page in main memory. Also called physical address.

Pool

A set of files created and maintained by a pool boss. More than one user number can access a pool.

Pool File

An ownership category that indicates a file can be accessed by any privileged task and by any task running under a user number the pool boss authorizes.

Private File

An ownership category that indicates a file can be accessed only by a task running under the user number under which the file is stored.

Privileged

A status that allows access to all files in the system and to most operating system functions. Contrast with Nonprivileged.

Problem Program

Virtual code file initiated either by the batch processor or from an interactive terminal.

Public File

An ownership category that indicates a file can be accessed by all users.

Scratch

A management category that indicates a file is to be destroyed upon termination of the task that created it.

Security Level

A level 0 through 7 established when a file is created. A user number is also associated with a security level. A user cannot access a file with a higher security level.

Small Page

512 contiguous 64-bit words.

Source File

A generic term for a file containing information used by a utility or other task whose specific meaning depends on the context of its use: the controllee file associated with a drop file, for instance, is termed the source file.

Also, in an UPDATE utility context, a file produced by UPDATE that would allow recreation of a new program library on a subsequent creation run.

Suffix

A letter A, B, C, or D that is associated with a task executing under a user number to differentiate from other tasks executing under the same user number. All batch jobs execute under suffix D; interactive tasks execute under the suffix specified at initiation of interactive processing.

System Billing Unit (SBU)

An installation defined unit used for charging of system resource usage. The unit might incorporate tape use access, number of tape functions, number of disk accesses, number of pages transferred to or from disk, and CPU usage in microseconds. An example of SBU is the time is microseconds of CPU use. See System Time Unit.

System Message

The means by which the operating system and user tasks communicate with each other. System messages are calls to the virtual system.

System Time Unit (STU)

An installation-defined unit used for allocating system resources. The unit might incorporate tape use/access, number of tape functions, number of disk accesses, number of pages transferred to or from disk, and CPU usage in microseconds. An example of STU is time in microseconds of CPU use. See System Billing Unit.

Task

An executable program.

User Number

Six digits that identify a file owner or user of system resources. Only one task can be in execution for a given user number and suffix at one time.

USER1

A generic name for privileged system tasks that process card decks, print files, and punch files. USER1 files are currently CARDREDR, PRINTOUT, PUNCHOUT, POSTOUT, SMP, CYBERIN, and CYBEROUT.

Virtual Address

Address that refers to virtual memory and is translated, through the page table, into a physical address.

Virtual Address Space

The set of virtual addresses that belong to a specific active task.

Virtual Code File

A file type that indicates an executable file having a minus page as its first page and a page zero as its second page. The file must be created by the loader. A virtual code file is also called a controllee file. Contrast with Object Code File.

Virtual Data File

A file type that indicates a nonexecutable file having a minus page as its first page and containing nonexecutable data only.

Virtual Memory

A concept by which physical main memory can be addressed as if it were as large as needed.

Virtual Range

Range of virtual addresses. Same as Virtual Address Space.

Word

A division of main memory or mass storage corresponding to 64 bits. Bits are numbered 0 through 63, left to right.

Working Set

Portion of a task's virtual address space that is referenced most frequently during execution of the task.

Working Set Size

Number of 512-word blocks in the working set.

Write-Temporary

A management category that indicates a file is read-only in mass storage. Modifications to the file are retained in the drop file and do not alter the write-temporary file image.

The CYBER 200 operating system assumes some conventions regarding the handling of the register file, an area containing 256 registers numbered from #0 to #FF. Some of the registers are used by the operating system for specific purposes, and others are available solely for the purposes of the user. One register file area of particular importance is the register save area, which is saved and restored each time an external procedure call is made.

REGISTERS

The register file is subdivided into six major areas, as shown in figure D-1. The environment register area and the temporary and working register area are jointly referred to as the register save area.

MACHINE REGISTERS

These registers include registers #0, #1, and #2. Register #0 contains machine zero (machine zero is described in the CYBER 200 Computer Hardware Reference Manual). Register #1 contains the data flag branch exit address, and register #2 contains the data flag branch entry address.

TEMPORARY REGISTERS

A user program can utilize two areas for temporary storage, addresses, or data. The two areas are from registers #3 to #13, and from #20 to the end of the register save area.

The lower area (#3 to #13) is large enough for execution of short subroutines (such as SIN and COS) completely within the temporary space, obviating the need for saving and restoring the register save area when short modules are needed by a program. However, some assemblers and compilers use this area; user caution is advised. The upper area (#20 to an upper limit specified by the caller), which is large enough to hold a variety of user procedures, is normally saved by the callee.

An error code from a station is returned by the system in register #FC. The possible values for this are defined in the resident call descriptions given in internal documentation.

GLOBAL REGISTERS

The contents of the global registers are universal to all programs within a specific execution/language system. The contents can be assumed by all modules within a given system and are not usually loaded, saved, or restored by called modules. The values in these registers are unique to a given operating environment; thus, if a module from a different environment is to be called, it is the caller's responsibility to establish the correct values for the callee in the proper registers.

The global registers and their contents are as follows:

Register #14 contains the constant #20.

Register #15 contains the constant #1A.

Register #16 contains the constant #1.

Register #17 contains the parameter descriptor. The number of parameters being passed during a call is contained in the leftmost 16 bits; the virtual bit address of the parameter list is stored in the rightmost 48 bits. Figure D-2 illustrates how parameters are passed to routines. If the address portion is zero, the parameters, if any, are being passed through the register file.

Registers #18 and #19 contain function results obtained from a called function. For example, the result of a trigonometric or exponential function would be placed in register #18. Register #19 could be used when a result has two components (for example, the imaginary part of a complex number whose real part is returned to register #18).

ENVIRONMENT REGISTERS

The environment registers consist of the minimum set needed to support the sharing of code in a virtual system and the general requirements of recursive, reentrant execution. These registers, along with a number of working registers, are the register save area. Each time a caller releases control to a callee, a new environment register/working area is established for the callee. A stack structure is used for this. The environment registers include:

#1A Return Register. Contains the virtual bit address of the location in the caller's program to which the callee's program normally returns.

#1B Dynamic Stack Pointer. Contains the relative bit base address of the next available free location in the dynamic stack. The address in this register must be a double-word address whenever a swap instruction (opcode 7D) is executed. The dynamic stack pointer is always advanced prior to storing data into that region or before addresses pointing to that region are calculated.

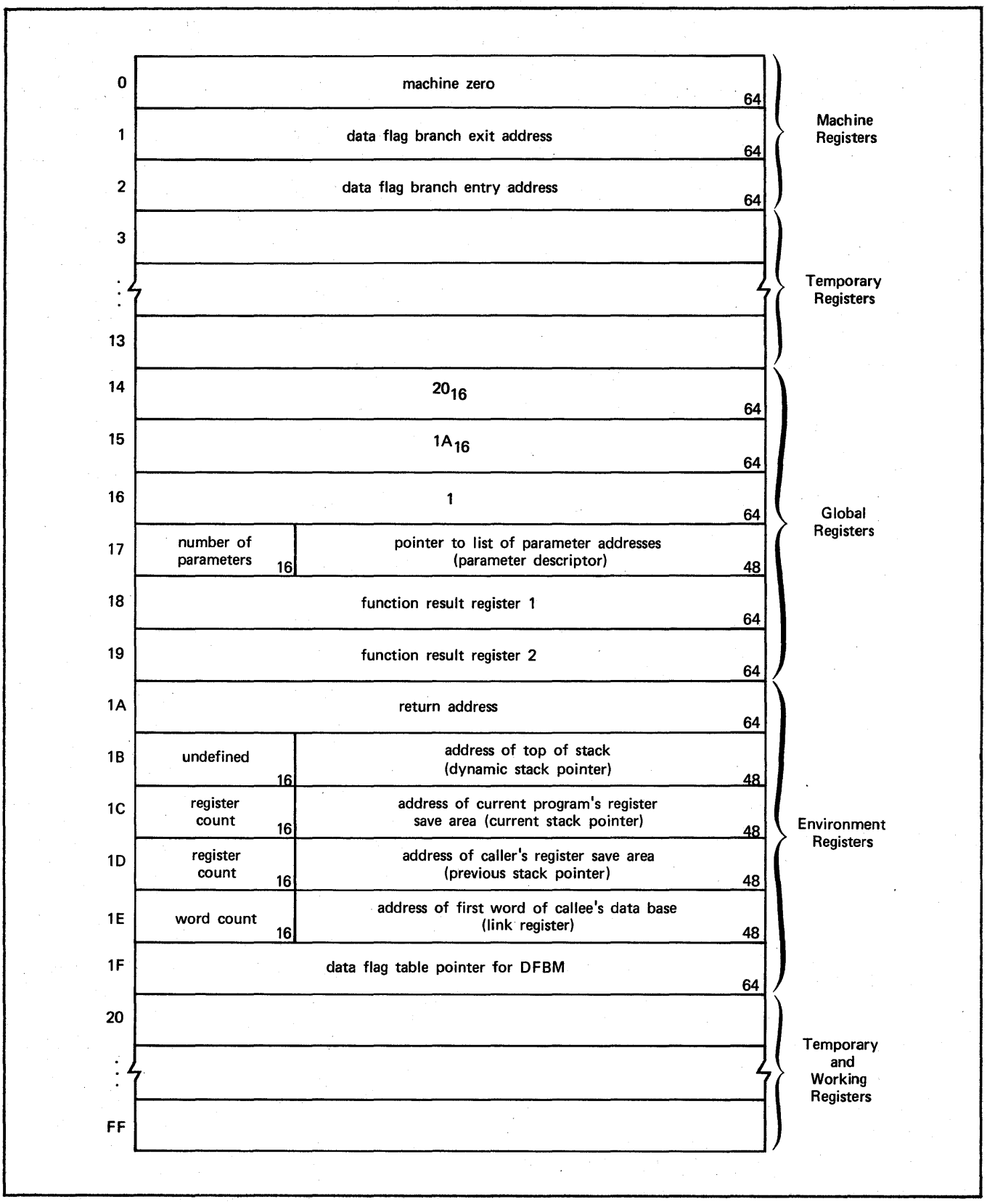


Figure D-1. Rigister File

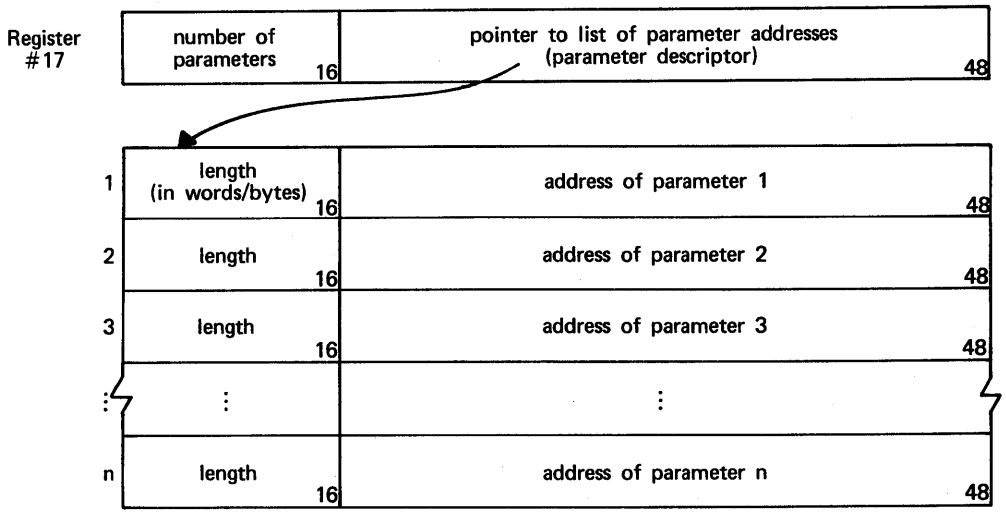


Figure D-2. Register #17

#1C Current Stack Pointer. Contains the length and relative bit base address of the region (the stack frame) in the dynamic stack where a caller wants his or her register save area to be saved. The minimum length of that region is the number of environment registers plus the number of registers (possibly none) needed for dynamic working storage for the program. During call sequences, the caller sets the length portion of the current stack pointer to the number of registers to be saved by the callee. The current stack pointer is set by the caller, but it is the callee that establishes the new stack frame. A minimum of six registers can be saved (the number of environment registers).

#1D Previous Stack Pointer. Contains the number of registers and the relative bit base address in the register file where the caller's register save area has been saved. The callee's previous stack pointer is an exact copy of the caller's current stack pointer.

#1E Link Register. Contains the length and virtual bit address of the data base allocated to the module by the loader. The caller passes to the callee the address of the callee's data base in the link register.

#1F Pointer to the data flag table for the data flag branch manager (DFBM).

The environment registers are used in two areas of a program, the prologue and the epilogue. The prologue and epilogue are generated by the assembler or compiler during assembly or compilation to ensure that the caller's register save area is saved when an external routine is called.

THE REGISTER SAVE AREA

The register save area is the space in which a program should store the permanent variables and addresses that it wishes to keep untouched whenever callee programs execute. When an executing program has called an external program, the instructions of the conventional prologue of the called program save the caller's register save area. The register save area is stored and saved as an element of a conventional chained stack in the register file. A stack element, called a stack frame, is diagrammed in figure D-3.

The initial size of a frame, defined by the difference of the values of the current stack pointer (the stack frame base) and the dynamic stack pointer, does not include temporary workspace. Any time temporary workspace is needed, the program can increment the dynamic stack pointer and in this way obtain space. Dynamic space use increases (frames are pushed onto the stack) until the lowest level called program has been executed; then, as the returns are encountered, the space is made available again in reverse order to the calls (frames are popped from the stack).

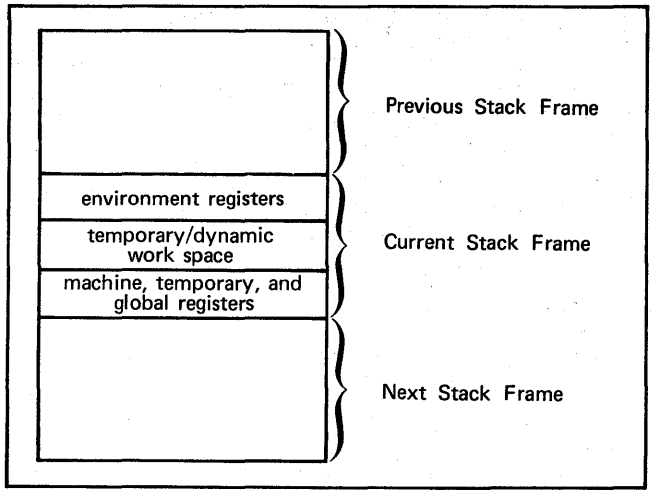


Figure D-3. Stack Frame

EXTERNAL PROCEDURE CALL SEQUENCE

The standard sequence of an external procedure call is as follows:

78xx001E	Load the link register with the address of the callee data base (xx = ep+1).
78yy0017	Load the parameter descriptor register.
361A00zz	Branch to the entry point of the called procedure and set a return location (contained in the return register).

In the above instructions, #17, #1E, and #1A are the register numbers of the parameter descriptor register, the link register, and the return register; xx contains the callee data base address, yy contains a descriptor of the parameter list, and zz contains the procedure entry point address.

PROLOGUE SEQUENCE

The prologue of the called procedure includes the following functions:

7D00151C	Starting with the register number specified by register #15 (#1A), save the registers specified in the stack of the caller.
2A1F0000	Clear the length part of word IF in the register save area for the data flag branch manager. (This instruction is required only when the data flag branch manager is used.)
781C001D	Place the current stack pointer in the previous stack pointer.
781B001C	Place the dynamic stack pointer in the current stack pointer.
2A1Cxxxx	Set the number of registers to be saved by the called program (xxxx = number of working registers of the callee plus six environment registers).
3F1Byyyy	Increment the dynamic stack pointer by 64 times the number of callee registers to be saved (yyyy = xxxx*64 bits); that is, set the dynamic stack pointer to the next free location.
2A1Ezzzz	Enter for zzzz the number of registers to be loaded for callee execution.
7D1E1400	Load the register file with data from the callee's data area in static space, starting with the register number (#20) specified by register #14.

Some programs can perform their subroutines entirely within the temporary registers, and do not make external calls. Such routines need not contain a prologue and can be assembled or compiled to omit it.

EPILOGUE SEQUENCE

The epilogue of the called procedure should be as follows; however, instructions 7E1F0005 through 3B060000 are required only when using the data flag branch manager:

7D1D1500	Using the length and address of the previous stack pointer, restore the register file from the callee's current stack, starting with register #1A (the environment registers).
7E1F0005	Load word zero from the data flag table to which register #1F points (ON_UNIT).
BE03180100000800	Enter the data flag register constant with the SFT, JIT, BKP, and enable bits set.
3B030004	Load and store the data flag branch register.
BE03180100000FE0	Enter the data flag register constant for an AND operation, which insures that previously set free and monitor flags remain set.
2D030406	Perform a logical conjunction of the current data flag register with the constant for free flags and monitor flags.

2E060506	Perform a logical disjunction of the current data flag register plus any free or monitor flags with word zero from the data flag table containing the data flag settings for the caller.
3B060000	Load the data flag register with the setting.
3340001A	Jump to the return address specified in register #1A.

A nonstandard return is carried out in similar fashion, except that the values of the previous stack pointer (#1D) and the return registers (#1A) are obtained from known variables.

The invisible package is a hardware convention that contains the address and control information required to begin a new job or to continue a job that was interrupted during execution. Each job is associated with an invisible package. When the CPU switches from monitor mode to job mode, the invisible package for the corresponding job is automatically loaded from main memory, beginning at the address assigned by the monitor. The invisible package data is loaded into the appropriate registers in the CPU. When the CPU switches from the job mode back to the monitor mode, as in the case of an interrupt, the contents of the corresponding registers are automatically stored in main memory as the invisible package for that job.

The contents of the invisible package are shown in figure E-1. X denotes a variable address bit. Because the fixed portion of the absolute word address is divided within the hexadecimal character, bits 52 through 55 are shown as their binary equivalents.

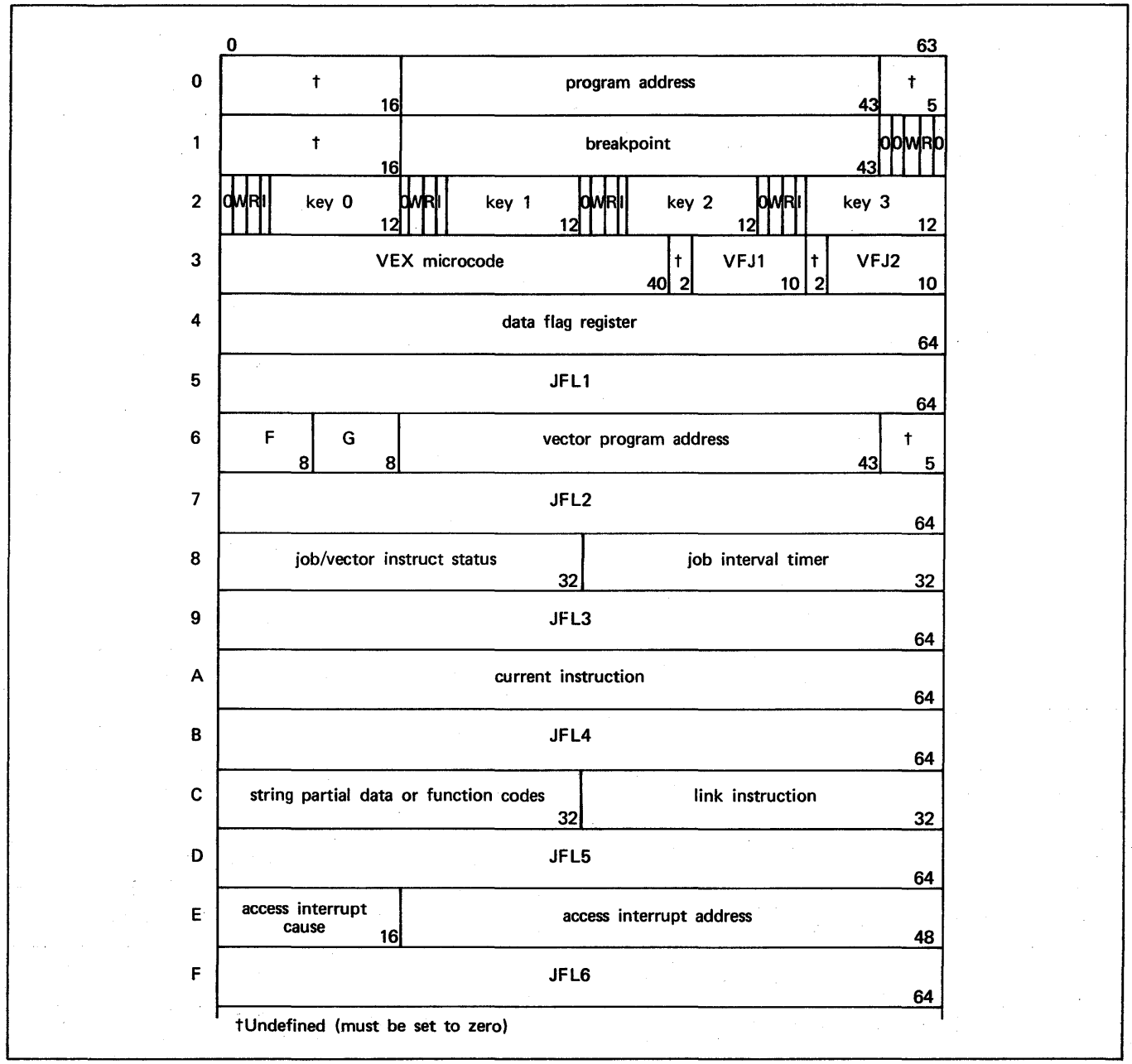
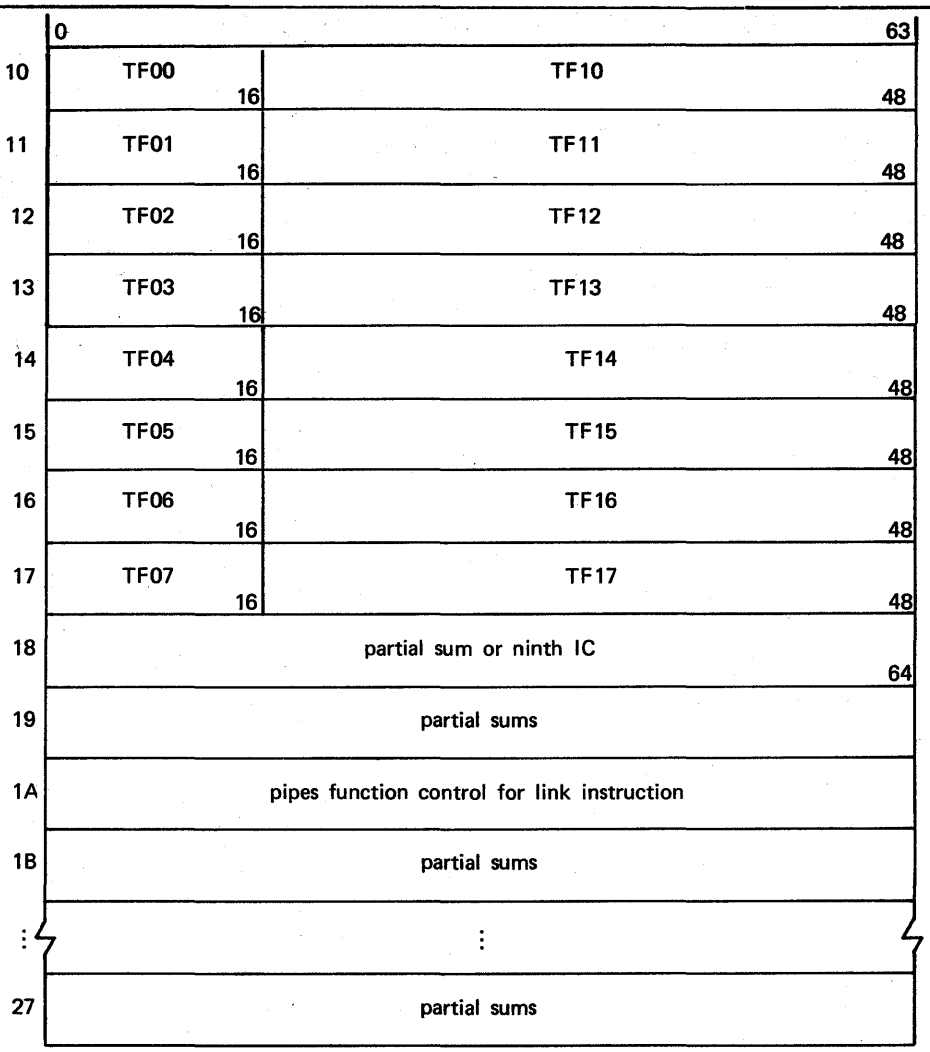


Figure E-1. Invisible Package Contents (Sheet 1 of 3)



Word 1 breakpoint usage bits:

0	Not used and must be zero
W	Check for breakpoint compare on write operands
R	Check for breakpoint compare on read operands

Word 2 usage lockout bits for each key:

0 Bits 0 and 16 together define a small page size for all small pages; bits 32 and 48 are not used and are set to zero

Bits 0 and 16

0	0	All small pages are 512 words
0	1	Undefined
1	0	All small pages are 2048 words
1	1	All small pages are 8192 words

W	Lockout CPU write operations
R	Lockout CPU read operations
I	Lockout CPU instruction references

Figure E-1. Invisible Package Contents (Sheet 2 of 3)

Word 3 VEX microcode conditions:

Bits 0 to 31	Not used and must be set to zero
32	Interrupt FF (signal to pipes)
33	Link instruction in execution
34	Link instruction R bit 3
35	Link instruction R bit 4
36	CC instruction in execution
37	Not used and must be set to zero
38	Vector block scalar use of Load/Store registers
39	Flag 1

Word 8 job/vector instruct status bits:

0	Vector restart
1	Not parallel operation
2 to 11	Undefined and must be set to zero
12	Stall bit (set for no data processed)
13	D8 or D9 execution started
14	Undefined and must be set to zero
15	EBCDIC when set, ASCII when clear
16	SCR Code Bit 3 (exit at vector instruction termination)
17	Select force of extension field length
18 and 19	Vector instruction reg file update disable bits
20	D8 or D9 multiple match flag
21	String restart bit (old data flag)
22 to 25	Undefined and must be set to zero
26	R-record FF
27	DA-DC Toggle Code Bit 0
28	DA-DC Toggle Code Bit 1
29	DA-DC Toggle Code Bit 2
30 and 31	Undefined and must be set to zero

Word C link instruction codes:

Bits 0 to 15	Link (56) instruction F and R codes
Bits 16 to 31	Link F1 instruction F and G codes

Word E access interrupt cause bits:

0 to 11	Not used and must be set to zero
12	Associative word not in page table
13	Write operand violation attempted
14	Read operand violation attempted
15	Read instruction violation attempted

Word 18 partial sum or ninth IC:

Bits 0 to 63	Partial sum for DX instruction or special broadcast quantity for link or CC instruction
Bits 0 to 15	Output item count for AX or C8 to CB instructions
Bits 16 to 63	C base address for AX instructions

These bits are undefined in all other applications.

Figure E-1. Invisible Package Contents (Sheet 3 of 3)

The current disposition of a program is indicated by a number that is carried in the descriptor block for the program. This code can be fetched by the privileged EXECUTE OPERATOR COMMAND message (f=0021).

Codes having specific definitions are shown in table F-1. Codes not defined in the table have these general meanings:

#0 to #9	Program is in the alternator loop.
#A to #F	Program is not in the alternator loop but is partially in memory.
#10 to #1F	Program is not processing a message and is waiting.
#20 to #2F	Program is processing a message and is waiting.
#30 to #3F	System is performing functions for a program.
#40 to #4F	Miscellaneous.
#B9 to #BF	State is indicated by subtracting #80. Tasks in a terminal or nonterminal dump state have #80 added to their original state when they are being dumped to disk; for example, #3D + #80 = #BD.

TABLE F-1. PROGRAM STATE CODES

Code	Meaning	Code	Meaning
0	Descriptor block (DB) is free	27	Waiting to send a message to the controllee
1	Program put in an alternator slot from the descriptor block load queue	28	(reserved for installation use)
5	Program alternator unblocked after new slot time	29	Waiting to send a message to the operator
7	Outstanding explicit input/output requests and interrupts have completed	2A	Waiting to send a message to the teletypewriter
11	Waiting for an alternator slot	30	Execute line in, descriptor block and keys assigned; message sent to load file management
13	Waiting for entry in the explicit input/output buffer	39	Terminate and kill all pages
14	Waiting for the mainframe	3A	Nonterminal dump. Suspend in state = #41 after completion.
15	Waiting for 7-track tape assignment	3B	Program dump, accounting finished; clean-up done; code + 80_{16} accounting done; dump input/output
16	Waiting for 9-track tape assignment		
1E	Waiting for the controllee to get on disk	3C	Dump finished, clean up to go
1F	Waiting for the controllee to get on disk	3D	Terminal dump error
20	Waiting for a message from the controller	3E	Terminal dump scheduled, no error
21	Waiting for a message from the controllee	3F	Nonterminal dump scheduled
22	(reserved for installation use)	40	Suspend for a time period
23	Waiting for a message from the operator	41	Suspended by the operator

INDEX

Abnormal termination control 4-56
Absolute address 1-1
Access station v; 1-2; C-1
Accessing magnetic tape files 4-3
Account identifier 3-1; C-1
Accounting 7-1
    System messages 4-2
Accumulating statistics 7-2
Active accounting file 7-1,2
ADVISE message 4-94
*AF file 7-2
Alpha word conventions 4-1
Alternator table 1-1,2
ANALYZER 6-1
ATC (refer to Abnormal termination control)
ATTACH PERMANENT FILE message 4-40
Attached private file 3-2
AUTOCON 7-9

Batch job C-1
Batch processor C-1
Beta word conventions 4-1
Bound explicit map 2-12
    Directory 2-7
    Entry format 2-15
Bound implicit map 2-12; C-1
    Directory 2-7
    Entry format 2-15
Breakpoint 5-1
Byte C-1

Calculating
    SBUs 7-9
    STUs 7-1
Call sequence D-4
CARDREDR 1-2
Central operating system 1-1
Central processing unit 1-1
CHANGE FILE ATTRIBUTES message 3-1; 4-31
CHANGE operator command 1-2
Changing
    Accounting rate 4-86
    File characteristics 4-31
    Memory limits 4-98
Character set A-1
Checkpoint C-1
CLOSE FILE message 3-2; 4-18
Closing files 4-18
Code block table 9-2
Code relocation table 9-3
Communication between tasks 4-2
Continuation lines 8-1
Controllee 2-7; C-1
    Chain C-1
        Processing 4-2
    Execute line 8-1
    File C-1
Controller C-1
Conventions for applications 8-1
Copying system tables 4-25
Core dump 6-1

CPU C-1
CPUQ 1-2
CREATE FILE message 3-3; 4-3
CREATE USER operator command 1-2
Creating a mass storage file 4-3
Cumulative accounting buffer 7-1,2
CYBER 200 Link Station 4-80; C-1
CYBER 200 OS 1-1
CYBERIN 1-2
CYBEROUT 1-2
C3xx resident calls 1-2
C5xx calls 1-3

Data base C-1
Data flag branch manager pointer D-3
Data item formats 9-5
Dayfile C-1
Dead codes B-1
Debug commands 5-3
Debug symbol table 9-9
Debugging tools 5-1
Descriptor block 1-1,2
    Number C-1
DESTROY FILE system message 3-1,3
Determining system access mode 8-1
DFBM pointer D-3
Diagnostics B-1
Disabling abnormal termination control 4-56
Disconnecting
    Controllees 4-52
    Tape files 4-9
Discontinuous virtual address ranges 2-12
Disk file accounting 7-4
Division 3-1; 7-1
Drop file 1-2; 3-2; C-1
    Map 2-12; C-1
        Entry format 2-16
        Format 2-16
Dump analysis 6-1
Dynamic stack pointer D-1

ENABLE/DISABLE ABNORMAL TERMINATION CONTROL message 4-56
Environment registers D-1
Epilogue D-3
    Sequence D-4
Error codes B-1
    For I/O operations 4-93
EXECUTE IQM REQUEST message 4-88
Execute line 8-1
    Special characters 8-12
EXECUTE OPERATOR COMMAND message 1-2; 4-58
EXECUTE PROGRAM FOR USER NUMBER message 4-70.7
Exit force instruction 1-1; 4-1
Explicit input/output 2-7; 3-2; C-2
EXPLICIT I/O message 1-3; 3-2; 4-88.2
External characteristics 2-1
External/entry table 9-3
    Format 9-4
External procedure call sequence D-4

Family file C-2
fi field 1-2
File C-2
  Access 3-1
  Concepts 3-1
  Extendability 1-3
  Management
    Categories 3-2
    System messages 4-2
  Map 2-7
  Name 3-1
  Ownership 3-1
  Space allocation 4-2
File index table 1-2; 2-1; 3-1; C-2
  Format 2-2
File segment table 1-3
Free space 1-1
  Attachments 2-12
FST (refer to File segment table)

GET MESSAGE FROM CONTROLLEE message 4-48
GET MESSAGE FROM CONTROLLER OR OPERATOR message 4-47
GET PACK LABEL AND PFI message 4-4
  Format 4-41
GIVE system message 3-1; 4-23
GIVE TAPE ACCESS TO CONTROLLEE system message 3-3; 4-35
GIVE UP CPU UNTIL I/O COMPLETES message 4-94
  Format 4-96
Global registers D-1

Hardware modes 1-1
Hexadecimal conversion tables A-1

Implicit input/output 3-2; C-2
INITIALIZE CONTROLLEE CHAIN message 4-55
INITIALIZE OR DISCONNECT CONTROLLEE message 4-52
Initiating program execution 4-2
Input queue entry format 4-70.5
Input queue manager 4-23
Input/output connection block 2-7
Input/output connector 2-7; C-2
  Format
    For link files 2-13
    For mass storage file opened for explicit input/output 2-9
    For mass storage file opened for implicit input/output 2-11
    For tape files 2-13
Internal characteristics 2-1
Interpretive data initialization table 9-5
Interpretive relocation initialization table 9-9
Interrupt 1-1
  Processing 4-94
    System messages 4-2
Interrupt address stack 2-8
  Format 2-14
Interrupt routines 4-94
Interrupting programs 4-54
Invisible package 2-7; C-2
  Format E-1
I/O operation error codes 4-93
I/O operation system messages 4-2
ioc (refer to Input/output connector)
IQM 1-2

Job accounting 7-9
Job management tables 2-1
Job mode 1-1

KERNEL 1-1,2
Key-dependent parameter format 8-2

Labeled tape C-2
Large page C-2
Left-hand-side table (refer to lhs table)
Level 4-54,94; C-2
lhs table 8-4
Library C-2
Line printer files 3-2
Link file creation parameters 4-8.1
LINK message 4-80
Link registers D-3
Link station v; 1-2
LIST CONTROLLEE CHAIN 4-42
LIST FILE INDEX OR SYSTEM TABLE message 4-22
LIST UNFORMATTED FILE INDEX TABLE message 4-22
Listing a controllee chain 4-42
Loader
  Conventions 9-1
  Table formats 9-1
Loader header table format 9-1
Loading files 2-7
Local files 3-2; C-2
Lockout 3-1; C-2
Logging in a mainframe 4-80
Logging out a mainframe 4-80

Machine registers D-1
Magnetic tape files (refer to Tape files)
Main memory C-2
Map directories 2-7
Map directory format 2-14
MAP message 3-1,2; 4-16
Mapping in a file 4-16
Mapping out a file 4-16
Mass storage file 3-2; C-2
Master clock entries 7-3
Memory access interrupts 1-1
Memory allocation 1-1
Memory dump analysis 6-1
Message communication 1-1
MESSAGE CONTROL message 4-50
Microdrum 1-4
Minus page 1-1; 2-7; C-2
  File maps 2-12
  Format 2-8
MISCELLANEOUS message 4-76
Modifying system table 1-1,2
Module header table format 9-2
Monitor mode 1-1

Nonprivileged users 3-1; C-2
NOS 1 v
NOS/BE 1 v

Object code file C-2
Object module format 9-2
OLE C-3

OPEN FILE message 1-3; 3-3; 4-10
Opening files 4-10
Operator command execution 4-58
OPERATOR system routine 1-2
Order-dependent parameter format 8-2
Order-independent parameter format 8-1
Output files 3-2; C-3

Pack File Index C-3
Page C-3
Page faults 1-2
Page swapping 1-1
PAGER 1-1,2
Paging in C-3
Paging out C-3
Parameter formats 8-1,2
Patterning released mass storage space 4-9
Periodic virtual system tasks 1-1
Peripheral operating system 1-1,4
Permanent files C-3
Permission 3-1
PFI C-3
Physical file 2-7; 3-2; C-3
Pool boss 3-1
POOL FILE MANAGER message 3-1; 4-79
Pool files 3-1; C-3
POSTOUT 1-2
Print files 3-2; 4-23
PRINTOUT 1-2
Private files 3-1; C-3
Privilege flag 1-2
Privileged C-3
  Resident system calls 1-1,3
  System tasks 1-1,2
  User numbers 1-2
  Users 3-1
  Virtual system calls 1-4
Problem program C-3
PROCESS SYSTEM PARAMETER message 4-98
Processing execute line 8-1
PROGRAM INTERRUPT message 4-54
Program states F-1
Prologue D-3
  Sequence D-4
Prompting 8-1
Pseudo address vector table 9-9,11
Public files 3-1; C-3
Punch files 3-2
PUNCHOUT 1-2

Queuing 1-1
Q5VRF file 7-9,10
Q7KEYWRD routine 8-1,3
  Call format 8-5
Q7MODE routine 8-1
  Call format 8-3
Q7PROMPT routine 8-1
  Call format 8-3

Read-only file access 3-1
RECALL message 4-79
Redirecting messages 4-50
REDUCE FILE LENGTH 4-25
Reducing mass storage file length 4-25
Register file
  Conventions D-1
  Format D-2
Register save area D-3
Releasing
  Mass storage space 4-9
  Tape drives 4-9
REMOVE CONTROLLEE FROM MAIN MEMORY message 4-50
Repositories 4-71; 7-1
Resetting breakpoints 5-4
Resident system 1-1
  Calls 1-3
Resource usage statistics 7-1
Retrieving
  Accounting statistics 4-38,76
  Formatted system table copy 4-25
  Pack label and file index 4-40
  Unformatted file index table entry 4-22
Return buffer 8-9
RETURN FROM INTERRUPT message 4-94
rhs table 8-4
Right-hand-side table (refer to rhs table)
ROUTE AND FILE DISPOSITION system message 3-2; 4-36
Routing files 4-36

SAE 7-9
Save table 4-51
SBU 7-9; C-4
Scan bits 1-4
SCANNER 1-4
Scanning text lines 8-3
Scheduler 1-1,2
Scratch files 3-2; C-3
Security levels C-3
Segmented files 1-3
Segments 3-1
SEND A MESSAGE TO CONTROLLEE message 4-44
SEND A MESSAGE TO CONTROLLER message 4-44
SEND A MESSAGE TO OPERATOR message 4-51
SEND MESSAGE TO DAYFILE message 4-86
Service level factors 7-9
Setting breakpoints 5-4
SIL subroutines 4-1
Small page C-3
SMP 1-2; 4-3
Source file C-3
Stack frame D-4
Stack pointers D-1
Standard processing 8-1
Standardized accounting enhancements (refer to SAE)
Station channel flags 1-1
Station IDs 4-80
Station message processor (refer to SMP)
Station processors 1-1
Statistics accumulation 7-2
STU 7-1; C-4
Subchannel number 4-80
Suffix C-3
Swapping controllee to mass storage 4-50
Symbol definition table 9-9,10
System
  Accounting 7-8
  Dead codes B-1
  Error codes B-1
  Library routines 8-1
  Table modification 1-1,2
System billing unit (refer to SBU)
System message 1-1; 4-1; C-4
  Function codes 4-2
System time unit (refer to STU)

Tape accounting 7-8
Tape files 3-3
Task C-4
Accounting 7-4
Name 8-1
TCHARGE routine 7-1
TELL DB operator command 4-51
Terminal accounting 7-4
TERMINATE system message 3-2; 4-21
Terminating execution 4-21
Time-slicing 1-1
T_JCAT system table format 4-70.6
Tokens 8-1
Transfer symbol table 9-9
Transferring file ownership 4-23
T_VRF table 7-9
T_VSD table 5-1
Structure 5-2

udtrust field 1-2
UPDATE USER DIRECTORY message 1-2; 4-71
User directory modification 1-2
User number 3-1; C-4
000000 3-1
USER1 routine 1-2; 4-23; C-4
USER/ACCOUNTING COMMUNICATION message 4-38; 7-2.1

Variable rate accounting 7-9
VARIABLE RATE ACCOUNTING message 4-86
Virtual address C-4
Space C-4
Virtual data file C-4
Virtual files 2-7; 3-3
Virtual paging 1-1
Virtual space requirements 4-94
Virtual system 1-1
Calls 1-3,4; 4-1
Tasks 1-1
Virtual system debug tool (refer to VSDT)
VRA (refer to Variable rate accounting)
VSDT 5-1
Commands 5-3

Word C-4
Working set 1-2; C-4
Write-only file access 3-1
Write-temporary files 3-2

△ xi
xi
*AF file 7-2

COMMENT SHEET

MANUAL TITLE: CDC CYBER 200 Operating System
Version 1 Reference Manual, Volume 2

PUBLICATION NO.: 60457010 **REVISION:** C

NAME: _____

COMPANY: _____

STREET ADDRESS: _____

CITY: _____ **STATE:** _____ **ZIP CODE:** _____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

☐ **Please Reply** ☐ **No Reply Necessary**

CUT ALONG LINE

PRINTED IN U.S.A.

AA3419 REV 1/80

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE. POSTAL REGULATIONS PROHIBIT THE USE OF STAPLES.

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

FOLD FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION